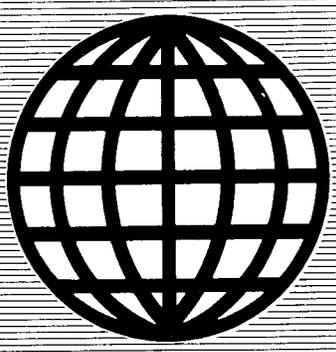


The COMPUTER JOURNAL®

Programming - User Support
Applications

Issue Number 32 $3.00

Language Development
Automatic Generation of Parsers for Interactive Systems

Designing Operating Systems
A ROM Based O.S. for the Z81

Advanced CP/M
Boosting Performance

Systematic Elimination of MS-DOS Files
Part 1 — Deleting Root Directories & an In-Depth look at the FCB

WordStar 4.0 on Generic MS-DOS Systems
Patching for ASCII Terminal Based Systems

K-OS ONE and the SAGE 68000
Part 2 — System Layout and Hardware Configuration

ISSN # 0748-9331

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, Montana
59912
406-257-9119

Editor/Publisher
Art Carlson

Art Director
Donna Carlson

Production Assistant
Judie Overbeek

Contributing Editors
Joe Bartel
Bob Blum
Bill Kibler
Rick Lehrbaum
Bridger Mitchell
Jay Sage

The Lillipute Z-Node sysop has made his BBS systems available to the TCJ subscribers. Log in on both systems (312-649-1730 & 312-664-1730), and leave a message for SYSOP requesting TCJ access.

Entire contents copyright© 1988 by The Computer Journal.

Subscription rates—$16 one year (6 issues), or $28 two years (12 issues) in the U.S., $22 one year in Canada and Mexico, and $24 (surface) for one year in other countries. All funds must be in US dollars on a US bank.

Send subscriptions, renewals, or address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, Montana, 59912, or The Computer Journal, PO Box 1697, Kalispell, MT 59903.

Address all editorial and advertising inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912 phone (406) 257-9119.

The COMPUTER JOURNAL

Features

Issue Number 32

Language Development

Parsing is a very important tool for command processing and parsing the input for a calculator is an easily understood example.
by Paul Mann 7

Designing Operating Systems

Some of the steps and considerations involved in designing a ROM based operating system.
by Clark Calkins 18

Advanced CP/M

Boosting system performance with fast disk resets, and optimizing sieve performance with tightly coded assembly language.
by Bridger Mitchell 20

Systematic Elimination of MS-DOS Files

An in-depth look at how the FCB works in conjuction with DOS, plus a program to delete root directory entries.
by Dr. Edwin Thall 26

WordStar 4.0 on Generic MS-DOS Systems

Version 4.0 can be patched to work with ASCII terminals for non-compatible systems (such as my AMPRO L.B. '186) or with remote terminals.
by Phil Hess 32

K-OS ONE and the SAGE 68000

Evaluating how the hardware design affects the BIOS requirements, and establishing the software flow diagrams.
by Bill Kibler 34

Columns

Editorial 3
Reader's Feedback 5
ZCPR3 Corner *by Jay Sage* 10
Computer Corner *by Bill Kibler* 44

Z sets you free!

Who we are

Echelon is a unique company, oriented exclusively toward your CP/M-compatible computer. Echelon offers top quality software at extremely low prices; customers are overwhelmed at the amount of software they recieve when buying our products. For example, the Z-Com product comes with approximately 92 utility programs; and our TERM III communications package runs to a full megabyte of files. This is real value for your software dollar.

ZCPR 3.3

Echelon is famous for our operating systems products. ZCPR3, our CP/M enhancement, was written by a software professional who wanted to add features normally found in minicomputer and mainframe operating systems to his home computer. He succeeded wonderfully, and ZCPR3 has become the environment of choice for "power" CP/M-compatible users. Add the fine-tuning and enhancements of the now-available ZCPR 3.3 to the original ZCPR 3.0, and the result is truly flexible modern software technology, surpassing any disk operating system on the market today. Get our catalog for more information - there's four pages of discussion regarding ZCPR3, explaining the benefits available to you by using it.

Z-System

Z-System is Echelon's complete disk operating system, which includes ZCPR3 and ZRDOS. It is a complete 100% compatible replacement for CP/M 2.2. ZRDOS adds even more utility programs, and has the nice feature of no need to warm boot (^C) after changing a disk. Hard disk users can take advantage of ZRDOS "archive" status file handling to make incremental backup fast and easy. Because ZRDOS is written to take full advantage of the Z80, it executes faster than ordinary CP/M and can improve your system's performance by up to 10%.

Installing ZCPR3/Z-System

Echelon offers ZCPR3/Z-System in many different forms. For $49 you get the complete source code to ZCPR3 and the installation files. However, this takes some experience with assembly language programming to get running, as you must perform the installation yourself.

For users who are not qualified in assembly language programming, Echelon offers our "auto-install" products. Z-Com is our 100% complete Z-System which even a monkey can install, because it installs itself. We offer a money-back guarantee if it doesn't install properly on your system. Z-Com includes many interesting utility programs, like UNERASE, MENU, VFILER, and much more.

Echelon also offers "bootable" disks for some CP/M computers, which require absolutely no installation, and are capable of reconfiguration to change ZCPR3's memory requirements. Bootable disks are available for Kaypro Z80 and Morrow MD3 computers.

Z80 Turbo Modula-2

We are proud to offer the finest high-level language programming environment available for CP/M-compatible machines. Our Turbo Modula-2 package was created by a famous language developer, and allows you to create your own programs using the latest technology in computer languages - Modula-2. This package includes full-screen editor, compiler, linker, menu shell, library manager, installation program, module library, the 552 page user's guide, and more. Everything needed to produce useful programs is included.

"Turbo Modula-2 is fast...[Sieve benchmark] runs almost three times as fast as the same program compiled by Turbo Pascal...Turbo Modula-2 is well documented...Turbo's librarian is excellent". - Micro Cornucopia #35

BGii (Backgrounder 2)

BGii adds a new dimension to your Z-System or CP/M 2.2 computer system by creating a "non-concurrent multitasking extension" to your operating system. This means that you can actually have two programs active in your machine, one or both "suspended", and one currently executing. You may then swap back and forth between tasks as you see fit. For example, you can suspend your telecommunications session with a remote computer to compose a message with your full-screen editor. Or suspend your spreadsheet to look up information in your database. This is very handy in an office environment, where constant interruption of your work is to be expected. It's a significant enhancement to Z-System and an enormous enhancement to CP/M.

BGii adds much more than this swap capability. There's a background print spooler, keyboard "macro key" generator, built-in calculator, screen dump, the capability of cutting and pasting text between programs, and a host of other features.

For best results, we recommend BGii be used only on systems with hard disk or RAMdisk.

JetFind

A string search utility is indispensible for people who have built up a large collection of documents. Think of how difficult it could be to find the document to "Mr. Smith" in your collection of 500 files. Unless you have a string search utility, the only option is to examine them manually, one by one.

JetFind is a powerful string search utility which works under any CP/M-compatible operating system. It can search for strings in text files of all sorts - straight ASCII, WordStar, library (.LBR) file members, "squeezed" files, and "crunched" files. JetFind is very smart and very fast, faster than any other string searcher on the market or in the public domain (we know, we tested them).

Software Update Service

We were suprised when sales of our Software Update Service (SUS) subscriptions far exceeded expectations. SUS is intended for our customers who don't have easy access to our Z-Node network of remote access systems. At least nine times per year, we mail a disk of software collected from Z-Node Central to you. This covers non-proprietary programs and files discussed in our Z-NEWS newsletter. You can subscribe for one year, six months, or purchase individual SUS disks.

There's More

We couldn't fit all Echelon has to offer on a single page (you can see how small this typeface is already!). We haven't begun to talk about the many additional software packages and publications we offer. Send in the coupon below and just check the "Requesting Catalog" box for more information.

Item	Name	Price	
1	ZCPR3 Core Installation Package	$49.00	(3 disks)
2	ZCPR3 Utilities Package	$89.00	(10 disks)
5	Z-Com (Auto-Install Complete Z-System)	$119.00	(5 disks) *
6	Z-Com "Bare Minimum"	$69.95	(1 disks)
10	BGii Backgrounder 2	$75.00	(2 disks)
12	PUBLIC ZRDOS Plus (by itself)	$59.50	(1 disk)
13	Kaypro Z-System Bootable Disk	$69.95	(3 disks)
14	Morrow MD3 Z-System Bootable Disk	$69.95	(2 disks)
16	QUICK-TASK Realtime Executive	$249.00	(3 disks)
17	DateStamper file time/date stamping	$49.95	(1 disk)
18	Software Update Service	$85.00	(1 yr sub)
20	ZAS/ZLINK Macro Assembler and Linker	$69.00	(1 disk)
21	ZDM Debugger for 8080/Z80/ HD64180 CPU's	$50.00	(1 disk)
22	Translators for Assembler Source code	$51.00	(1 disk)
23	REVAS3/4 Disassembler	$90.00	(1 disk)
24	Special Items 20 through 23	$169.00	(4 disks)
25	DSD-80 Full Screen Debugger	$129.95	(1 disk)
27	The Libraries.SYSLIB, Z3LIB, and VLIB	$99.00	(8 disks)
28	Graphics and Windows Libraries	$49.00	(1 disk)
29	Special Items 27, 28, and 82	$149.00	(9 disks)
30	Z80 Turbo Modula-2 Language System	$89.95	(1 disk)
40	Input/Output Recorder IOP (I/OR)	$39.95	(1 disk)
41	Background Printer IOP (BPrinter)	$39.95	(1 disk)
44	NuKey Key Redefiner IOP	$39.95	(1 disk)
45	Special Items 40 through 44	$89.95	(3 disks)
60	DISCAT Disk cataloging system	$39.99	(1 disk)
61	TERM3 Communications System	$99.00	(6 disks)
64	Z-Msg Message Handling System	$99.00	(1 disk)
66	JetFind String Search Utility	$49.95	(1 disk)
81	ZCPR3: The Manual bound, 350 pages	$19.95	
82	ZCPR3: The Libraries 310 pages	$29.95	
83	Z-NEWS Newsletter, 1 yr subscription	$24.00	
84	ZCPR3 and IOPs 50 pages	$9.95	
85	ZRDOS Programmer's Manual 35 pages	$8.95	
88	Z-System User's Guide 80 page tutorial	$14.95	

* Includes ZCPR3: The Manual

Echelon, Inc.

P.O. Box 705001-800
South Lake Tahoe, CA 95705
(916) 577-1105

NAME _____

ADDRESS _____

_____

TELEPHONE _____ DISK FORMAT _____

☐ REQUESTING CATALOG

ORDER FORM

Payment to be made by:
- ☐ Cash
- ☐ Check
- ☐ Money Order
- ☐ UPS COD
- ☐ Mastercard/Visa:

_____

Exp. Date _____

California residents add 7% sales tax.
Add $4.00 shipping/handling in North America, actual cost elsewhere.

ITEM	PRICE
_____	_____
_____	_____
_____	_____
Subtotal	_____
Sales Tax	_____
Shipping/Handling	_____
Total	_____

Editor's Page

An Idea Whose Time Has Come?

In a previous editorial (*The HELL With Being Compatible, I Want What I Want!* issue #30 page 4), I offered the outrageous idea of using a stripped down Operating System containing only the features needed by a program, and then reinstalling the regular system upon program completion. In that same issue, Jay sage discussed NZCOM, which included some of the same approaches. Now, in this issue, Jay Sage unveils the greatly expanded NZCOM.

Like so many things whose time has come, these ideas were developed simultaneously by several different people. NZCOM has much wider applications than what I had planned, and it will be much more useful to the general programming community. I encourage you to acquire and use both NZCOM and ZCPR.

My implementation (if I ever finish it) will be specifically tailored for the program with which it runs. I am currently planning on using a floppy which will boot and run the dedicated program/system, making use of the hard drive if necessary. I want the operator to be able to simply insert the disk and hit the reset, and then be up and running with out ever seeing the prompt or having to enter any commands.

The way most people use hard drives is OK for experts who know their way around the directories and subdirectories, or for single use dedicated applications where the initial boot can place them in the correct area, configured and ready to run. It is not so good where non-expert operators are expected to run several different programs, in different areas, which require different configuration. Using a boot floppy can automatically place them in the desired program, in the correct area, with the correct configuration. I feel that non-experts should never see the system prompt, and I'm not completely comfortable with shell programs which require a menu selection to initialize the operation.

Small Software Companies Are in Trouble

We contact a lot of computer software companies in connection with a direct mail promotion campaign, and about half of them indicate that they are in financial trouble.

Some of their comments are, "We're reevaluating our products," "All of our sales and marketing will be turned over to a distributor," "Our budget is zero," or "We have no plans for any additional advertising." Sometimes we call back and find the phones disconnected (even while ads are still running). Today I found that two companies which I talked to in the past 14 days have had their phones disconnected.

Making it with a small business has always been a difficult proposition, but it is especially difficult for a small software company at this time. There are more computers being used than ever before, but most of them are being used by non-technical people who require (and expect) the support of a large organization. The consumers are also impressed by expensive multi-media promotions, and good products by small companies are being forced out of the market by large companies with huge advertising budgets. I like Borland's Turbo Pascal, and I think that their Turbo C will help introduce a lot of people to C programming. I'm uncomfortable with the fact that Turbo C is forcing some other good C products out of the market. Just as the IBM PC became the small computer standard, Turbo C is becoming the de facto C standard. Regardless of how good a product is, I would like to see a number of other good products coexisting in the market with friendly competition driving all them to provide improvements in product performance and user support.

The only advice that I have to offer small software developers is to very carefully define your market, and to look for unconventional methods of promoting your product. You have to produce orders (not just inquires) at an acceptable cost, and the expensive mass media high hype campaigns don't work for small distribution high tech products. I could provide several articles on this if there is enough interest — give me some feedback if you are interested.

Seagate ST-225 Problems

There has been a lot of talk about ST-225 failures. I had one installed in an AMPRO Bookshelf® unit, and after 2 to 3 months it started to occasionally fail to boot or give the dreaded BDOS error. I found that the ST-225s run VERY HOT!! I moved the system to a TeleVideo 806/20 box with lots of room and a good fan. I even mounted the ST-225 on standoffs so that air could circulate on all sides. The same drive which previoulsy gave trouble has now operated for 3 months with no problems. I'm not sure that this is a sure fix for drives which have been damaged because of excessive heat problems, but I certainly would not mount a ST-225 in a standard PC type box. ST-225s run HOT!! If I get any NEW ones at a bargain price, I'll mount them in a separate box with lots of space and lots of air (and then back up frequently).

I'm looking forward to the day when we can economically install 20 to 30 megabytes of non-volatile RAM and do away with mechanical drives except for archive backups.

Data Base Program Source Code

I realize that I sound like a chronic complainer, but I can't help it that none of the commercial programs satisfy me. Many of them do amazing things which I

don't need, but fail to do what I want the way I want it done. I used to wonder why people still use custom programmers when there are so many off-the-shelf programs available — now I understand that it is beause the canned programs don't satisfy the business needs.

My current frustration is with setting up a large data base to handle up to 100,000 names for a mailing list. The requirements are simple, as it is a flat file (a simple one-file base with no relational requirements). The two DBMS programs I use (Condor and Clipper (which is a dBase III syntax compiler)) perform most of the required data base functions, but there are some operations which are much easier to accomplish with assembly language, C, or Pascal. For example, I want to use a binary bit encoded field in each record to keep track of where that record was used. Then I can read the record and check the code field with a bit mask to determine if it should be used in the current job. If it isn't used I'll just read the next record; but if it is used, I'll set a bit to indicate this use and write the record out to a file (or send it to the printer). So far I haven't discovered how to handle Hex code and binary bit logic from the DBMS program.

I also want a separate data entry program which I can supply to individuals for data entry, which means that I have to be able to provide copies without licensing restrictions. The data entry program will have to provide for the data validation that I want, such as a legitimate two character state code, and a five digit ZIP code which is valid for that state. I'll probably also automatically break the data into separate disk files based on the first digit of the ZIP code in order to limit the disk file size (don't keep all of your eggs in one basket in case of a crash).

I don't want to write an entire DBMS program, but I do want to work around the restrictions I find in the available programs. The best solution appears to be to use the DBMS programs where they work well, and supplement them with custom programs where needed. I am looking for more input on this, and will welcome any thoughts or source code you have to share. If we receive enough material we can establish a regular column and make the code available as user disks.

Language Development

There is a lot of activity in language development, about which we hear very little — and there should be even more work done in this area. I'm not talking about the large companies such as MicroSoft, Borland, and Digital Research, but rather smaller companies which are developing specialized languages for their own purposes. The developments which I hear about seem to be concentrated in the area of control and automation, but I'm sure that there is also a lot of work being done in other areas — I just haven't asked the right questions in the right places.

When someone talks about developing a language, we automatically think about the widely known major projects such as Prolog, Modula, or Turbo C® . These projects are intended for wide distribution, and involve millions of dollars and tens of man years. The little known languages are more modest, but still very important, developments which are intended for a more limited specific application.

An example of a little known language is STD BASIC® from Octagon Systems, 6510 W. 91st Avenue, Westminster, CO 80030 phone (303) 426-8540. This was developed for their line of STD Bus products, and includes statements and functions such as: AIN which returns the result of an A/D conversion at a port address, FREQ which returns the frequency of a periodic port signal, TACH which returns the equivalent RPM of an I/O port signal, DEV which returns the Standard Deviation of an array, MAXPOS which returns the array position of the maximum value, and STORE which transfers a program in RAM to EPROM, plus many more.

While it is true that any good programmer could write a program to accomplish these tasks, Octagon provides a modified BASIC which (in conjunction with their boards) allows nonprogrammers to develop control programs. Forth was developed for machine control, but it is much easier to get a nonprogrammer started doing something useful in a specialized BASIC than it is to get them started in Forth. Paul Mann (LALR Research, 1892 Burnt Mill, Tustin, CA) reports that some of his parser customers are developing specialized languages.

What should a specialized language look like? What is the difference between a language and a program which parses the user's input and acts on it? I'm very interested in parsing and language development, and would like to hear from anyone doing work in this area.

Magazine for Hardware Hackers

We recently received the premier issue (January/February) of *Circuit Cellar Ink*, and the Editorial Director is the well known Steve Ciarcia. This issue includes: The Circuit Cellar Motion Triggered Video Camera Multiplexer, High Security on a Budget — Build a Video Handscanner/Identifier, The Home Satellite Weather Center — Part 1: RGBI to NTSC Converter; plus several departments.

It is published bimonthly by Circuit Cellar Inc., 4 Park Street, Suite 12, Vernon, CT 06066 (203-875-2751), for $14.95 per year. Write to Steve at this address to let him know what you'd like to see in future issues

I had heard that Byte would be dropping Steve's Circuit Cellar — does this mean that it's true? ■

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these registered trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used marks are acknowledged, and we apologize for any we have overlooked.

Apple II, II+, IIc, IIe, Lisa, Macintosh, DOS 3.3, ProDos; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. DateStamper, BackGrounder ii, DosDisk; Plu*Perfect Systems; Clipper, Nantucket; Nantucket, Inc. dBase, dBase II, dBase III, dBase III Plus; Ashton-Tate, Inc. MBASIC, MS-DOS; Microsoft. WordStar; MicroPro International Corp. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C; Borland International. HD64180; Hitachi America, Ltd. SB180 Micromint, Inc.

Where these, and other, terms are used in The Computer Journal, they are acknowledged to be the property of the respective companies even if not specifically acknowledged in each occurrence.

Reader's Feedback

820 Hacker

I am presently using/hacking a Xerox 820-1. I have been hacking around with the 820-1 for close to three years now and I have done quite a few interesting things with it. If I can get a little extra time I would like to submit some of the upgrades and additions that I have added to the Xerox, some of which are listed below:

- Changing the WS1771 disk controller to a WD2795.
- Installing the Andratech EPROM Programmer.
- Monitor ROM enhancements.
- Building a keyboard translator ROM.
- Adding a capital key lock to a parallel keyboard (some keyboards only have a shift lock key).
- Increasing the stepping rate from 12mS to 3mS.
- CRC-16 generation.
- 2.5 to 4.0 mHz upgrade.
- Fast disk copy program.
- Centronics printer interface.
- General hardware: Z80, PIO, SIO, etc.

In addition to all present columns in TCJ, I would enjoy the following:

- I'd like to see a "LOOK WHAT I FOUND" column. This column could be sent in by a user each issue who has come across an interesting public domain software program.
- I'd like to see a "SOMEONE TELL ME" column each month that explains how to do a particular thing that a user has requester in a previous issue.
- Z80/64180 assembly language column. How to do this and that . . .
- Turbo Pascal column.
- C programming column.
- Word Processing column — WordStar 4.0?
- MicroMint SB-180 column.
- Articles on Disassemblers — Z80DIS for example.
- Articles on Debuggers — Z8E for example.
- Articles on basics of CP/M: BDOS calls, BIOS calls, the I/O byte, etc.
- ZCPR articles (enjoy Jay Sage's articles).
- Hardware articles: what's a PIO, SIO, CTC, etc.
- 68000 topics.

G.B.

Big Board User

My personal systems are two Big Boards (much expanded) and two S-100 systems. All have 8″ drives and CP/M 2.2. At the office I have a Zenith PC-158 (2 drives + HD) and an Ampro Bookcase CP/M computer with 2 5¼ (750K drives). I maintain some 10+ Zeniths and miscellaeous Macs for the school as well as all the scientific instruments (X-ray, electron microscope, mass spec., etc.).

I am interested in hardware, special projects, do it yourself operating systems, as well as small special purpose computers, as that is what I do for money. I can do assembly programming but do not "work as a programmer" as it is just to make the stuff I design and build for some one operate as it should, i.e. when I make a small special purpose computer I write the code for it but that is about all now.

CP/M still lives (at home). Keep up the spirit

J.D.

---

8 Inch Drive Problems

Can any one tell me where I can obtain an 8″ alignment disk?

Thomas M. Butler
3015 Linwood Avenue
Parkville, MD 21234
(301) 665-3927

Miscellaneous Reader Comments

I use a Zenith Z-248 with Seagate 40 Meg Hard Drive and two 360K floppies. I am interested in data acquisition systems, signal processing and graphics. Keep up the good work in presenting excellent hardware and software articles in your publication.

K.S.

Using and "OLD" (dosen't seem that old) Heath 89. Mostly like subjects where I will have a fair understanding of the inner workings, software and hardware. The Journal, therefore is great!

V.V.

Running H89 with three drives, one 8″ and two 5¼ 80 tracks. Also a SB180 (MicroMint) using the H89 as the terminal. Using CP/M Magnolia on the H89, and Z System on the SB180.

Would like to see an article on using two computers with ont terminal.

G.S.

Two CP/M machines at home (Kaypro 8 & 10), both using ZCPR2.5. It's a shame that ZCPR3 is such a pig on TPA.

I use MS-DOS, VMS(VAX) and Macintosh at work, plus a SUN (UNIX).

I like CP/M. Need articles on the Centronics parallel bus.

W.P.

My current system is a surplus Televideo 860 (Z80 CP/M), but my dream system is a VMEbus 680X0.

I'd like to see 68K projects, 68K BBSs, VME bus information. Ed Scott was doing a series on a 68K/VME project for *Computer Smyth*. Now that CS is defunct, perhaps TCJ could pick that up? You need a column on OS/9.

I like your series on SCSI, and the Hawthorne 68K board.

G.B.

Ampro L.B. & SCSI

I have to let you know how greatly appreciated you publication is. The value received greatly outweighs the price paid. Thanks.

I am using the Ampro L.B. Z80 w/SCSI harddrive, NSystems Ramdisk, Kenmore clock, and the Heritage terminal card. Am very pleased with it. Recently had to hunt down a problem with SCSI bus, not knowing if the problem was with the controller or SCSI chip/host. The articles by Rick Lehrbaum helped me to debug to the point that showed me the NCR 5380 wasn't handshaking. Couldn't have done without it.

I'm interested in more about SCSI (tape drives?) and 68000 info. Also highly hopeful in seeing the Z280 come out. Hope Zedux continues with its development.

C.W.

Apple

I use an Apple II system. How about an Apple II hardware article once in a while?

Do you know where I can get an Apple Lisa keyboard schematic?

Richard M. Ramsbey
25703 Layton Road
North Liberty, IN 46554

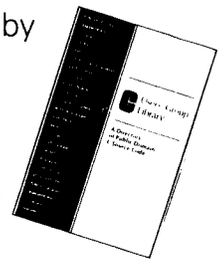
Editor's Note: Can any one out there help Richard with the Apple Lisa?

C Users' Group

Over 115 volumes of Public Domain C source code, including: editors, compilers, communication packages, text formatters, UNIX-like tools, etc., available in over 100 formats.

NEW! $10

CUG Directory of Public Domain C source code.
200+ pages of file by file descriptions and index.



Write or call for more details

The C Users' Group
Box 97 McPherson, KS 67460 (316) 241-1065

Turbo Pascal Advanced Applications

The ADVANCED reference for Turbo Pascal programmers. A must for anyone using Turbo Pascal. You will really charge up your programs and become an ***expert***.

Learn how to:

- Use the DOS background print spooler.
- Create libraries and use data compression techniques.
- Optimize your code, data structures, and I/O.
- Do command line processing.
- Make the most of available memory and break the 64K data limit.
- Use interrupts from Turbo Pascal.
- Install a DOS command processor for instant access to any program from within Turbo Pascal.
- Understand bit mapped graphics, and implement the Core graphics in Turbo Pascal.
- Build a subset Pascal compiler.

Pick the brains of 13 Turbo Pascal experts, each writing in his area of special interest.

A good blend of theory, practical applications, and lots of usable code.

Book includes MS DOS disk.

Price: $32.95 postpaid in USA. Foreign orders: add $2.00 for surface shipment. MC and VISA accepted.

ORDER FROM:

Rockland Publishing, Inc.
190 Sullivan, Suite 103
Columbia Falls, MT 59912
(406) 257-9119

Language Development

Automatic Generation of Parsers for Interactive Systems

by Paul Mann, LALR Research

Parsing, breaking down a command into its separate components, is a very essential element of interactive computer systems. Building a parser seems like a simple task at first, however, using the wrong approach leads to maintenance problems, performance problems and reliability problems.

This article demonstrates a good design technique when you have access to a parser generator such as the one available from LALR Research which was used for this article. A good parser is no longer a major expense, and a list of easily affordable parser generators for personal computers is located at the end of this article.

The principle of parsing is the same for calculators and command languages. However, a calculator is the better choice for illustration purposes. If you can specify the input to your system with a formal description, then your job is halfway done. If you can't specify the input in a formal manner, then you may not understand your input specification.

Since a calculator is a well known entity, a formal description of its input is well understood. A formal grammar for a simple calculator is shown in Figure 1.

```
Goal    -> Stmt <cr>
Stmt    ->
        -> Exp
        -> Target = Exp
        -> quit
Target  -> <identifier>
Exp     -> Term
        -> Exp + Term
        -> Exp - Term
Term    -> Factor
        -> Term * Factor
        -> Term / Factor
Factor  -> Primary
        -> - Primary
Primary -> ( Exp )
        -> <number>
        -> <identifier>
```

Figure 1. Expression grammar for a calculator.

A good parser generator can display the complete parsing action report which shows the finite states and all related state transitions. LALR 3.0 generates the listing shown in Figure 2 from the expression grammar above.

Each state contains the terminal transitions, nonterminal transitions and reductions required for processing the type of expressions described by this grammar. There is not enough room in this article to explain how the parser generator builds these states and actions, but the bibliography lists several compiler books which contain information on parsers.

All you need to know is that the parser generator can take the grammar and produce a parser capable of processing expressions. State 0 is the start state. "+=>" means accept the indicated symbol and goto the next state as indicated. "<=" means reduce, the indicated production (rule) has been recognized. The parser generator "plugs" these numbers into the parser skeleton provided and outputs a working parser in source code form. Then you compile the parser and link it into the rest of your system.

The parser interacts with the rest of your system at the appropriate times by calling the functions specified in the grammar. These function names are attached to rules in the grammar. The complete grammar needed by the parser generator is shown in Figure 3. It also contains some information that specifies the interface to the scanner.

```
STATE  0    SLR(1)
*    1  Stmt -> .
*    0  Goal -> . Stmt <eof>

     1  Stmt ........ +=>    6
     3  Exp ......... +=>    7
     2  Target ...... +=>    8
     4  Term ........ +=>    9
     5  Factor ...... +=>   10
     6  Primary ..... +=>   11
     8  quit ........ +=>    1
     1  <identifier>  +=>    2
    10  - ........... +=>    3
    13  ( ........... +=>    4
     4  <number> .... +=>    5
        DEFAULT ..... <=     1

STATE  1  quit  LR(0)
*    4  Stmt -> quit .

        DEFAULT ..... <=     4

STATE  2  <identifier>  SLR(1)
*    5  Target -> <identifier> .
*   16  Primary -> <identifier> .

     7  = ........... <=     5
        DEFAULT ..... <=    16
```

Figure 2. First three parser states and actions.

```
/* Terminal symbols coming from scanner. */

   <cr>                    /* Carriage Return. */
   <number>
   <identifier> => LOOKUP
   <operator>   => LOOKUP
   <punctuator> => LOOKUP

/* Productions. */

   Goal    -> Stmt <cr>
   Stmt    ->
           -> Exp              => DISPLAY
           -> Target = Exp     => STORE
           -> quit             => QUIT
   Target  -> <identifier>     => ADDR
   Exp     -> Term
           -> Exp + Term       => ADD
           -> Exp - Term       => SUB
   Term    -> Factor
           -> Term * Factor    => MUL
           -> Term / Factor    => DIV
   Factor  -> Primary
           -> - Primary        => NEG
   Primary -> ( Exp )
           -> <number>         => PUSH
           -> <identifier>     => RECALL

/* End. */
```

Figure 3. Complete grammar for interfacing with a scanner and the rest of the system.

LOOKUP is a function that looks up the incoming symbol in the table of valid grammar symbols and assigns a terminal symbol number. DISPLAY sends the result of the expression to the CRT. STORE stores the result in the symbol table location indicated by "Target". QUIT terminates the program. ADDR puts the symbols table address for the "Target" on the stack. ADD performs addition with the first two values on the stack. SUB performs subtraction. MUL performs multiplication. DIV performs division. NEW changes the sign of the first value on the stack. PUSH pushes the "<number>" on the stack. RECALL recalls the symbol table location for "<identifier>" and pushes it on the stack.

For the expression 1+2*(3+4) the moves of the parser are shown in Figure 4. Note that the result of the expression is left on the stack. The notation CR indicates a carriage return or end of line.

Some of the semantic actions are shown in Figure 5. You can see how simple they really are and how easy the actions were to specify in the grammar.

INPUT SYMBOL	PARSE ACTIONS	SYNTAX STACK	SEMANTIC ACTIONS	SEMANTIC STACK
1	shift	1		
+	reduce P <= 1	P	PUSH 1	1
+	reduce F <= P	F		1
+	reduce T <= F	T		1
+	reduce E <= T	E		1
+	shift	E+		1
2	shift	E+2		1
*	reduce P <= 2	E+P	PUSH 2	1 2
*	reduce F <= P	E+F		1 2
*	reduce T <= F	E+T		1 2
*	shift	E+T*		1 2
(	shift	E+T*(		1 2
3	shift	E+T*(3		1 2
+	reduce P <= 3	E+T*(P	PUSH 3	1 2 3
+	reduce F <= P	E+T*(F		1 2 3
+	reduce T <= F	E+T*(T		1 2 3
+	reduce E <= T	E+T*(E		1 2 3
+	shift	E+T*(E+		1 2 3
4	shift	E+T*(E+4		1 2 3
)	reduce P <= 4	E+T*(E+P	PUSH 4	1 2 3 4
)	reduce F <= P	E+T*(E+F		1 2 3 4
)	reduce T <= F	E+T*(E+T		1 2 3 4
)	reduce E <= E+T	E+T*(E	ADD	1 2 7
)	shift	E+T*(E)		1 2 7
CR	reduce F <= (E)	E+T*F		1 2 7
CR	reduce T <= T*F	E+T	MUL	1 14
CR	reduce E <= E+T	E	ADD	15

Figure 4. Moves of an LR parser for input 1+2*(3+4).

```
PUSH () /* Push value onto stack. */
{
        *(++StackP) = ATOF (T_beg);
}

ADD () /* Add top two values on stack and pop. */
{
        *(--StackP) = *StackP + *(StackP+1);
}

SUB () /* Subtract top two values on stack and pop. */
{
        *(--StackP) = *StackP - *(StackP+1);
}

MUL () /* Multiply top two values on stack and pop. */
{
        *(--StackP) = *StackP * *(StackP+1);
}

DIV () /* Divide top two values on stack and pop. */
{
        *(--StackP) = *StackP / *(StackP+1);
}

NEG () /* Negate top value on stack. */
{
        *StackP = - *StackP;
}
```

Figure 5. Some of the semantic action source code.

The generated parser automatically takes care of the parse stack. In the function PUSH the variable T_beg is a global variable defined in the parser that points the text containing the values '1', '2', '3', etc.

There's not enough room in this type of article to explain everything. So only a brief exposure can be given. In short, an LALR parser generator can be used effectively to build a calculator. By doing so a software engineer can: 1) save time, 2) produce reliable systems and 3) produce high performance systems.

Programming is a tedious, painstaking task. If some of that work can be done by a parser generator, then that leaves more time for designing and doing more creative work. ■

Sources for LALR(1) parser generators:

TWS
$6000
MetaWare, Inc.
903 Pacific Ave, Suite 201
Santa Cruz, CA
408-429-6382

QPARSER+
$300
QCAD Systems, Inc.
1164 Hyde Ave
San Jose, CA 95129
408-727-6884

LALR 3.0
$99
LALR Research
1892 Burnt Mill
Tustin, CA 92680
714-832-LALR

Bison
$25
Austin Code Works
11100 Leafwood Lane
Austin, TX 78750
512-258-0785

YACC
$10
C Users Group
P O Box 97
McPherson, KS 67460
316-241-1065

Bibliography:

1. "COMPILERS: PRINCIPLES, TECHNIQUES AND TOOLS," Aho, Sethi & Ullman, 1986, Addison Wesley.

2. "COMPILER CONSTRUCTION: THEORY AND PRACTICE," Second Edition, Barrett, Bates, Gustafson & Couch, 1985, Science Research Associates.

3. "INTRODUCTION TO COMPILER CONSTRUCTION WITH UNIX," Schreiner & Friedman, 1985, Prentice Hall.

4. "COMPILER CONSTRUCTION," Waite & Goos, 1984, Springer-Verlag.

5. "PRINCIPLES OF COMPILER DESIGN," Aho & Ullman, 1977, Addison Wesley.

C CODE FOR THE PC

source code, of course

Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
CQL Query System (SQL retrievals plus windows)	$325
GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
Barcode Generator (specify Code 39 (alphanumeric), Interleaved 2 of 5 (numeric), or UPC)	$300
Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$295
Aspen Software PC Curses (System V compatible, extensive documentation)	$250
Vitamin C (MacWindows)	$200
TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
Essential resident C (TSRify C programs, DOS shared libraries)	$165
Essential C Utility Library (400 useful C functions)	$160
Essential Communications Library (C functions for RS-232-based communication systems)	$160
Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$150
Greenleaf Functions (296 useful C functions, all DOS services)	$150
OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
American Software Resident-C (TSRify C programs)	$130
PC Curses Package (full System V, menu and data entry examples)	$120
CBTree (B+tree ISAM driver, multiple variable-length keys)	$115
Minix Operating System (U**x-like operating system, includes manual)	$105
PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
The Profiler (program execution profile tool)	$100
Entelekon C Function Library (screen, graphics, keyboard, string, printer, etc.)	$100
Entelekon Power Windows (menus, overlays, messages, alarms, file handling, etc.)	$100
Wendin Operating System Construction Kit or PCNX, PCVMS O/S Shells	$95
C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
Professional C Windows (windows and keyboard functions)	$80
JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
WKS Library (C program interface to Lotus 1-2-3 program & files)	$80
ME (programmer's editor with C-like macro language by Magma Software)	$75
Professional C Windows (lean & mean window and keyboard handler)	$70
Quincy (interactive C interpreter)	$60
EZ_ASM (assembly language macros bridging C and MASM)	$60
PTree (parse tree management)	$60
HELP! (pop-up help system builder)	$50
Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
Heap Expander (dynamic memory manager for expanded memory)	$50
Make (macros, all languages, built-in rules)	$50
Vector-to-Raster Conversion (stroke letters & Tektronix 4010 codes to bitmaps)	$50
Coder's Prolog (inference engine for use with C programs)	$45
C-Help (pop-up help for C programmers ... add your own notes)	$40
Biggerstaff's System Tools (multi-tasking window manager kit)	$40
CLIPS (rule-based expert system generator, Version 4.0)	$35
TELE Kernel or TELE Windows (Ken Berry's multi-tasking kernel & window package)	$30
Clisp (Lisp interpreter with extensive internals documentation)	$30
Translate Rules to C (YACC-like function generator for rule-based systems)	$30
6-Pack of Editors (six public domain editors for use, study & hacking)	$30
ICON (string and list processing language, Version 6 and update)	$25
LEX (lexical analyzer generator)	$25
Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor)	$25
AutoTrace (program tracer and memory trasher catcher)	$25
C Compiler Torture Test (checks a C compiler against K & R)	$20
Benchmark Package (C compiler, PC hardware, and Unix system)	$20
TN3270 (remote login to IBM VM/CMS as a 3270 terminal on a 3274 controller)	$20
A68 (68000 cross-assembler)	$20
List-Pac (C functions for lists, stacks, and queues)	$20
XLT Macro Processor (general purpose text translator)	$20
Data	
WordCruncher (text retrieval & document analysis program)	$275
DNA Sequences (GenBank 52.0 including fast similarity search program)	$150
Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$60
Webster's Second Dictionary (234,932 words)	$60
U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
USNO Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FidoNet: 1:382/12

Free shipping on prepaid orders **For delivery in Texas add 7%** **MasterCard/VISA**

The ZCPR3 Corner

by Jay Sage

As usual, I find myself with the deadline for this TCJ column fast approaching, wondering how two months have passed so quickly. And, as usual, I have far more material that I would like to talk about than I will have time to put down on paper (or, should I say, disk). This column is probably going to be shorter than average, just as the previous one was much longer, and some of the promised discussions will be deferred still further. Given the reasons, you will be understanding I hope: Joe Wright and I are in the process of putting the final touches on the new releases of ZCOM and ZCPR34! By the time you read this, they will definitely be available. Even if you usually find my columns a bit too technical for your tastes, I hope you will read on as I describe these two exciting developments.

I will not describe it here this time, but Bridger Mitchell has very nearly completed code similar to NZCOM that will run on CP/M-Plus systems. At last people with newer CP/M machines for which CP/M 2.2 is not available will also be able to run Z System. And they will be able to do it while retaining almost all of the good features of CP/M-Plus!

The New ZCOM

Two issues ago (TCJ #30) I described the status of the nascent NZCOM. Things have developed considerably since then, and I can now provide some specific details.

First some philosophical comments. This may sound rather strong, but Joe and I both firmly believe that NZCOM is one of the most exciting and remarkable developments in the history of microcomputer operating systems. With all the computers we have had experience with, the operating system has been a static entity. You 'boot' up the computer, and there you have the operating system, fixed and immutable. Few computers offer more than one operating system. With those that do, the only way you can get a different operating system is to 'reboot', which generally involves inserting a new boot diskette and pressing the reset button. And never do you get to define the characteristics of that operating system. You just take what the manufacturer deigns to let you use.

With NZCOM the operating system becomes a flexible tool just like an application program. You can change operating systems at any time, even right in the middle of a multiple command line sequence. You can do it manually, or alias scripts can do it automatically, in response to conditions in the system! And you can determine which Z System features are supported.

You can change the whole operating system or just a part of it. Would you like a new command processor? No problem. With a simple command, NZCOM will load it. No assembly or configuration required. One file fits all! That new CCP will continue to run until you load another one. Want to experiment with a new disk operating system (we are playing with several exciting new ones)? Again, no problem. NZCOM can load them in a jiffy. This makes for a whole new world of flexibility and adaptability, experimentation and education.

Need more memory to run a big application program? Fine. Just load a small operating system while that application is running. When it is finished, go back to the big system with bells and whistles like named directories, lots of resident commands, or special input/output facilities (such as keyboard redefiners or redirection of screen or printer output to disk).

Until you try this system, it is hard to imagine how easy it is to do these things. Gone are the days of taking source code (no source code is needed), editing configuration files (you don't need an editor), assembling (you don't need an assembler), and patching (you don't have to know how to use the arcane SYSGEN and DDT). Simple REL files for a particular module can be used by anyone on any system. Of course, if you want to create custom modules of your own special design, you can still do it, but this is no longer required, as it used to be. Hackers can hack, but users can simply use!

Joe and I are really hoping that NZCOM will open the world of Z System to the general community, to those who have no interest in learning to assemble their own operating system or do not have the tools or skills. If you have been at all intrigued by the Z System (how could you not have been?), now is your chance to experiment.

Getting NZCOM running is basically a two-step process, with each step remarkably easy to perform. First you define the system or systems you want. This is done with the program MKNZC (MaKe NZ-Com). Then you load the Z System you want using the program NZCOM. The details are explained below. Some comments of interest to the technically inclined are enclosed in brackets. Feel free to skip over them.

Defining NZCOM Systems

Here is how a person with a stock CP/M computer would go about getting NZCOM going. [First technical aside: Ironically, those of us who, with great skill and hard work, created manually installed Z Systems have a much harder job ahead of us. To use NZCOM effectively, we must first strip out all the ZCPR3 code in our fancy BIOSs and get back to a lean, Z-less system. I just spent the good part of an evening doing that for my BigBoard I computer (though, to be fair to my programming expertise, I should add that the hardest part was finding where I had stashed the BIOS source code).] For the discussion that follows, we will assume that the files in the NZCOM package have been copied onto a working disk in drive A.

As we said earlier, the first step is to use MKNZC, an easy menu-driven program, to specify the characteristics of our Z Systems. Its output is a descriptor file that is used later to load the system. What if you don't know enough yet about the Z System to make those choices? Again, no problem. There is a standard (or, in computer language, 'default') system defined for you already, and we will start by making it. We do that by entering the command line:

```
A>mknzc nzcom
```

This will bring up a menu screen like the one shown in Figure 1. The only difference on your system will be in the actual ad-

dresses for the modules, since they vary from computer to computer. Press the 'S' key to save the configuration. MKNZC displays a message to the effect that it is writing out the file NZCOM.NZC.

[Technical aside: Files of type NZC are NZCOM descriptor files. They are simple text files, as shown in Figure 2. For those of you who write your own assembly language programs, you may notice a strong similarity to the symbol or SYM file produced by an assembler or linker (yep, identical). The symbols in this file define all the necessary parameters of the system to be created.]

From the values in Figure 1, you can see that the default Z System offers every feature available. When this system is running later, the TPA (transient program area, the memory available for the application programs that do your real work) will be 49.0k bytes. This value, of course, is for my computer; as they say, "yours may vary." A 'k' or kilobyte is actually 1024 bytes, so this is really 50,176 bytes or characters. The original CP/M system, by the way, had a TPA of 54.25k bytes, so we are paying a cost of 5.25k bytes for this spare-no-expense Z System. As luxurious and opulent as this system is, it still leaves plenty of TPA for most application programs.

Sometimes, however, we have an application program that is really hungry for memory. Database managers, spread sheets, and C compilers often fall into this category. So does the new WordStar Release 4® . We will now use MKNZC to define a minimum Z System for when we run those applications. To give this version the name MINIMUM, enter the command:

A>mknzc minimum

When the menu comes up, press key '4'. You will be asked to define the number of records (128-byte blocks) to allocate to the input/output package or IOP. Enter '0' and press return. Similarly reduce to zero the allocations for the resident command package (RCP), flow command package (FCP), and named directories register (NDR). You will be left with the screen shown in Figure 3. Press the 'S' key to save the definition of this minimal Z System in the descriptor file MINIMUM.NZC [shown in Figure 4 for the technically inclined].

Notice that the TPA has grown to 53.25k, only 1k less than the original miserable CP/M system. Even with this meager Z System, costing only 1k of TPA, you get the following features (and more):

•multiple commands on a line
•the alias facility that provides automatic command sequence generation

```
1.*   Command Processor       CCP   BD00    16 Records
2.*   Disk Operating System   DOS   C500    28 Records
3.*   NZ-COM Bios             BIO   D300     2 Records

4.    In/Output Processor     IOP   D400    12 Records
5.    Resident Command Proc   RCP   DA00    16 Records
6.    Flow Control Processor  FCP   E200     4 Records
7.    Named Directory Reg     NDR   E400    14 Names

8.*   Environment Descriptor  ENV   E500     2 Records
9.*   Shell Stack             SHS   E600     4 Entries

P.    Custom Patch Area       PAT   0000     0 Records
      Customer's CBIOS        TOP   E800

              Effective TPA size 49.0k

* Items 1, 2, 3, 8 and 9 are not changeable in this version.

Selection: (or <S>ave or <Q>uit) _
```

Figure 1. Screen displayed by the MKNZC program when run under CP/M. This is the standard or default system definition.

```
E806 CBIOS    0080 ENVTYP   E6F4 EXPATH   0005 EXPATHS   DA00 RCP
0010 RCPS     D400 IOP      000C IOPS     E200 FCP       0004 FCPS
E400 Z3NDIR   000E Z3NDIRS  E700 Z3CL     00CB Z3CLS     E500 Z3ENV
0002 Z3ENVS   E600 SHSTK    0004 SHSTKS   0020 SHSIZE    E680 Z3MSG
E6D0 EXTFCB   E7D0 EXTSTK   0000 QUIET    E6FF Z3WHL     0004 SPEED
0010 MAXDRV   001F MAXUSR   0001 DUOK     0000 CRT       0000 PRT
0050 COLS     0018 ROWS     0016 LINS     FFFF DRVEC     0000 SPAR1
0050 PCOL     0042 PROW     003A PLIN     0001 FORM      0066 SPAR2
0042 SPAR3    003A SPAR4    0001 SPAR5    BD00 CCP       0010 CCPS
C500 DOS      001C DOSS     D300 BIO      0000 PUBDRV    0000 PUBUSR
```

Figure 2. For the technically inclined, this is a listing of the contents of the NZCOM.NZC system descriptor file produced by MKNZC.

```
1.*   Command Processor       CCP   CE00     16 Records
2.*   Disk Operating System   DOS   D600     28 Records
3.*   NZ-COM Bios             BIO   E400      2 Records

4.    In/Output Processor     IOP   0000      0 Records
5.    Resident Command Proc   RCP   0000      0 Records
6.    Flow Control Processor  FCP   0000      0 Records
7.    Named Directory Reg     NDR   0000      0 Names

8.*   Environment Descriptor  ENV   E500      2 Records
9.*   Shell Stack             SHS   E600      4 Entries

P.    Custom Patch Area       PAT   0000      0 Records
      Customer's CBIOS        TOP   E800

              Effective TPA size 53.25k

* Items 1, 2, 3, 8 and 9 are not changeable in this version.

Selection: (or <S>ave or <Q>uit) _
```

Figure 3. Screen displayed by the MKNZC program after eliminating the IOP, RCP, FCP, and NDR modules in order to define a minimal Z System.

•automatic, user-defined search path for COM files
•extended command processing (ARUNZ, described in TCJ #31, for example)
•error handling that tells you what's wrong with a bad command and allows you to correct it
•shells (menu systems, command history shell for saving and recalling old commands, file-maintenance shells, etc.)
•terminal-independent full-screen operation via Unix-like TCAP (terminal capabilities descriptor)

These are only two of a wide variety of possible Z Systems. As you gain experience with NZCOM, you can fine tune the definitions to meet all of your needs. For my BigBoard I computer, I have defined four systems. Two of them, called FULL and TINY, have the features shown in the

two examples here. A third one is called SMALL. Not quite as diminutive as TINY, it sacrifices an additional 0.5k of TPA to retain the flow command package (FCP), which is so valuable in providing high levels of command automation. Even my voracious application programs can usually get by under this system.

Finally, I have a system called NORMAL, which, as the name implies, is the one I use most of the time. It is the same as FULL but without an IOP. The most common use for an IOP is to run keyboard redefiners like NuKey. Most people like this feature, but splendid as NuKey is, for some reason my style does not find much use for keyboard macros (I've become a rather skillful typist and can generally type faster than I can think of moving my finger to a special key), so I generally omit the IOP and gain 1.5k of TPA.

```
E806 CBIOS      0080 ENVTYP     E6F4 EXPATH     0005 EXPATHS    0000 RCP
0000 RCPS       0000 IOP        0000 IOPS       0000 FCP        0000 FCPS
0000 Z3NDIR     0000 Z3NDIRS    E700 Z3CL       00CB Z3CLS      E500 Z3ENV
0002 Z3ENVS     E600 SHSTK      0004 SHSTKS     0020 SHSIZE     E680 Z3MSG
E6D0 EXTFCB     E7D0 EXTSTK     0000 QUIET      E6FF Z3WHL      0004 SPEED
0010 MAXDRV     001F MAXUSR     0001 DUOK       0000 CRT        0000 PRT
0050 COLS       0018 ROWS       0016 LINS       FFFF DRVEC      0000 SPAR1
0050 PCOL       0042 PROW       003A PLIN       0001 FORM       0066 SPAR2
0042 SPAR3      003A SPAR4      0001 SPAR5      CE00 CCP        0010 CCPS
D600 DOS        001C DOSS       E400 BIO        0000 PUBDRV     0000 PUBUSR
```

Figure 4. For the technically inclined, this is a listing of the file MINIMUM.NZC, which describes a minimum-size version of an NZCOM system for the computer in Figures 1 and 2.

```
A>nzcom /v
NZCOM Ver 2.0 Copyright (C) 1987-88 Alpha Systems Corp. 21 Jan 88
 Input buffer start 1C00
 Read  buffer start 1D00
 Write buffer start 3D00
 Loading A0:NZCOM.NZC
 Loading A0:NZCPR.REL for BD00 at 3D00
 Loading A0:NZDOS.REL for C500 at 4500
 Loading A0:NZBIO.REL for D300 at 5300
 Loading A0:NZIOP.REL for D400 at 5400
 Loading A0:NZRCP.REL for DA00 at 5A00
 Loading A0:NZFCP.REL for E200 at 6200
 Loading A0:NZCOM.NDR for E400 at 6400
 Loading A0:NZCOM.Z3T for E580 at 6580
 Writing A15:NZCOM.CCP
 Booting NZ-COM...
```

Figure 5. This is the screen display produced by NZCOM as it loads the default system definition NZCOM.NZC with the verbose option.

Loading the NZCOM Systems

Having defined the systems above, we can now fire them up even more easily. For the default NZCOM system, just enter the following simple command:

A>nzcom

With no argument after the command name, NZCOM will load the system defined with the name NZCOM. As it does this, you will see a signon message on the screen, followed by a series of dots, each one indicating that another module has been loaded. [Technical aside: If you want to see more precisely what is going on, just add the option '/v' to the command to select verbose mode. You will then get a screen display something like that shown in Figure 5. I'll have more to say about what all this means a little later.]

After NZCOM starts running, it executes a program called START.COM. This is usually an alias command, a program that simply passes another more complex command line on to the command processor. I will not explain the details of START here, but after it finishes, Z System will be up and running, waiting for your commands.

How NZCOM Works

This section is for the technically inclined, so if that's not you, pretend there are square brackets around this whole section and skip ahead to the next section. Here we are going to explain what some of those verbose-mode messages mean and what NZCOM is doing to create the system on the fly.

First NZCOM loads the descriptor file into memory. Among other things, this file has the information about which system modules to load and to what starting addresses. The first module loaded is the command processor. It is loaded from the file NZCPR.REL, which has the code for the command processor (ZCPR34) in so-called relocatable form.

There is some very interesting assembly/linkage razzle-dazzle that goes on here. With the REL files one usually plays with, only the run-time execution address of the code is unknown at assembly time and must be resolved by the linker. Things are much trickier here. When the command processor code was assembled, not only was its own run-time starting address unknown, but the addresses of various other system components, such as the message buffer and multiple command line, to which it refers in countless places, are also unknown. Since there is no fixed relationship between the addresses of the CCP and these other modules, there is no way to define the addresses using equates in the code.

Put another way, when NZCOM converts NZCPR.REL into actual object code, it must resolve not only the calls and jumps and data loads that refer to other locations in the command processor but also those that refer to the other system modules. Fortunately, advanced assemblers and linkers — including those from SLR Systems and a ZAS follow-on under development by Echelon — already have a mechanism to handle this problem. It was Bridger Mitchell who recognized how this mechanism, called named common, could accomplish what was needed here.

When code with symbols in named common is assembled, the corresponding bytes in the resulting REL file are marked not only for relocation but for relocation with respect to a specific common block. The SLR assemblers support up to 12 named common blocks. NZCOM contains very sophisticated linking code that resolves the references to data items in the common blocks, the addresses of which it gets, naturally, from the NZC descriptor file.

Figure 6 shows a partial listing of the file NZCMN.LIB, which is referenced in a MACLIB statement in each module assembled for use by NZCOM. Seven named common blocks are defined: __BIOS__, __ENV__, __SSTK__, __MSG__, __FCB__, __MCL__, and __XSTK__ for the CBIOS, environment descriptor, shell stack, message buffer, external file control block, multiple command line buffer, and external stack, respectively. Note that no common blocks are defined for the RCP, FCP, or NDR. References to these package must be made indirectly at run time, using data obtained from the environment descriptor in memory.

How does the NZCOM loader figure out that the file NZCPR.REL is the command processor? You might think that it uses the name of the file, but, in fact even if you had a copy of it called MYNEWCP.REL, NZCOM would be able to load it just as well. The answer is

```
; Named COMMON declarations start here
; For compatibility, these are the same names used by Bridger Mitchell's
; JetLDR

        common  /_BIOS_/
cbios:                          ; Customer's bios address

        common  /_ENV_/
z3env:                          ; Z3 Environment descriptor
z3envs  equ     2               ; Size (records)
rcp     equ     z3env+12
rcps    equ     yes             ; Used as existence test, not size
fcp     equ     z3env+18
fcps    equ     yes             ; Used as existence test, not size
z3ndir  equ     z3env+21
z3ndirs equ     yes             ; Used as existence test, not size

drvec   equ     z3env+52        ; Valid drive vector

ccp     equ     z3env+63        ; CCP entry
ccps    equ     z3env+65        ; Size

dos     equ     z3env+66        ; DOS entry (+6)
doss    equ     z3env+68        ; Size

bio     equ     z3env+69        ; BIO entry

        common  /_SSTK_/
shstk:                          ; Top of Shell stack
shstks  equ     4               ; 4 entries
shsize  equ     32              ; 32 bytes each

        common  /_MSG_/
z3msg:                          ; Message buffer
z3msgs  equ     80              ; 80 bytes long

        common  /_FCB_/
extfcb:                         ; External file control block
extfcbs equ     36              ; 36 bytes long
expath  equ     extfcb+extfcbs  ; External path
expaths equ     5               ; 5 elements
z3whl   equ     expath+(expaths*2)+1 ; The wheel byte
z3whls  equ     1               ; 1 byte

        common  /_MCL_/
z3cl:                           ; Multiple command line
z3cls   equ     203             ; Maximum command length
nzpat   equ     z3cl+256        ; Potential User patch area

        common  /_XSTK_/
extstk:                         ; External stack
extstks equ     48              ; Size (bytes)

        cseg                    ; Select Code Segment

; End of NZCMN.LIB
```

Figure 6. This is a partial listing of the file NZCMN.LIB, which defines the named common blocks used during assembly of modules for use by NZCOM.

```
B2:DBASE>nzcom minimum /v
NZCOM Ver 2.0 Copyright (C) 1987-88 Alpha Systems Corp. 21 Jan 88
 Input buffer start 1C00
 Read  buffer start 1D00
 Write buffer start 3D00
 Loading A0:MINIMUM.NZC
 Loading A0:NZCPR.REL for CE00 at 3D00
 Loading A0:NZDOS.REL for D600 at 4500
 Loading A0:NZBIO.REL for E400 at 5300
 Loading A0:NZCOM.Z3T for E580 at 5480
 Writing A15:NZCOM.CCP
 Booting NZ-COM...
```

Figure 7. This is the screen display when NZCOM loads the minimum system from a running default system.

that the source code contains the directive

NAME ('CCP')

which gives the REL file an internal module name. It is this name that NZCOM uses to determine what kind of module the code represents.

After the command processor is loaded, the other modules are loaded in succession in similar fashion, except for two. The named directory file NZCOM.NDR is a file that you can make or change with the standard utility programs MKDIR or EDITNDR/SAVENDR. There is nothing in an NDR file that requires relocation at all. The same is true for the Z3T terminal descriptor (TCAP) file. It can be created using the TCSELECT utility.

When all the loading is done, a copy of the command processor object code is written out to a file called NZCOM.CCP. This file is used for subsequent warm boots, since we obviously cannot warm boot from what is on the system tracks of the disk (the Digital Research command processor is still there, after all). At this point we can resume the non-technical discussion.

Changing NZCOM Systems

Now that you have Z System running, you can start to work with it and learn about it. I am not going to discuss Z System in general here; the subject is much too extensive. One thing you can do is to get out your back issues of TCJ and experiment with the programs described there. Another is to buy the *Z System User Guide* published by Echelon. That book describes the Z System from a less technical point of view than Richard Conn's *ZCPR3, The Manual,* also published by Echelon.

What I would like to discuss now is some of the ways you can use the dynamic capabilities of NZCOM. First we will describe how you change the entire operating system. For these examples we will assume that you have been doing work in various directory areas on your system and that you have set up named directories. Let's say you are in your dBase II® area now. Since you know that dBase II needs a lot of memory to run efficiently (or should I say 'tolerably,' since it never runs efficiently!) and since (unlike WordStar 4, for example) it cannot make use of any Z System features anyway, you want to load the minimum system we created earlier. You can probably guess what the command is:

B2:DBASE > nzcom minimum

[More technical stuff: Figure 7 shows the screen display you would get with the '/v' verbose option on this command.] For the minimum system NZCOM loads

only a command processor, disk operating system, and virtual BIOS. The other system segments disappear. This includes the NDR or named directory register, so the prompt changes to

B2>

The START alias does not run this time. It runs only when NZCOM is loaded from a non-NZCOM system (such as CP/M).

In general, when loading a new version of the operating system from another that is currently running, NZCOM loads only the modules that must be loaded, either (1) because they did not exist before or (2) because they are now at a different address or have a different size. For example, when I load my FULL system from the NORMAL system to add an IOP, only the CCP, DOS, BIOS, and IOP are loaded, since the RCP, FCP, and NDR are in the same place as before and have the same size. When modules do have to be loaded, files with the default names shown in Figure 5 are used. Later we will discuss how you can load modules with other names.

There are a number of system modules that never change in the present version of NZCOM. (Yes, like the famous Al Jolson lines, you ain't seen nothin' yet!) These include the environment descriptor, message buffer, shell stack, path, wheel byte, and multiple command line buffer. With the exception of module addresses in the environment descriptor, data in these fixed system modules remain unaffected. This means that if you had selected an error handler, for example, or a shell such as a command history shell, they will still be in effect after a change of system.

Because the multiple command line buffer is preserved through the load of a new system, you can include NZCOM commands as part of multiple command sequences, alias scripts, and shell (MENU, VMENU, or ZFILER) macros. Thus, for example, you could have entered the command

B2:DBASE>nzcom minimum;dbase etc.

In this case the operating system would have changed, and then DBASE would have started running. I will not go into the technical details here, but there are ways to write an alias script, which might be called DB, that would check to see if the minimum system was already running and, if not, automatically load it before invoking DBASE.

Nothing says the operating system can change only once in the course of a multiple command line. You might have alias scripts that change to a minimum system, run a specific command, and then reload the normal system again. There is a time penalty associated with this (though very little if you have the NZCOM files on a RAM disk), but the result is that the application program sees a big TPA while it is running, but **you** always see a nice, full-featured Z System.

NZCOM does not even insist that you stay in Z System. On the contrary. On a cold load from CP/M it will build (if it does not exist already) a program called NZCPM that, when run from Z System, will restore the original CP/M system.

[Technical aside: Even if you need absolutely every available byte of TPA, you can still automate the process. You can use the submit facility to run a batch job that exits from Z System entirely, runs an application under plain CP/M, and then returns to Z System. You do have to observe some precautions. For example, you have to make sure that all command lines in the batch file that will execute while Z System is not in effect are valid CP/M commands. Once the batch script has reloaded Z System using the NZCOM command, it can resume using appropriate Z System commands, including multiple commands on a line.

Another factor to bear in mind is that NZCPM returns you to CP/M in drive A user 0 no matter where you were when it executed. Since ZCPR3 (starting with version 3.3) writes its submit file to directory A0 rather than the current directory, there is no problem with continuing operation of the batch file under CP/M. However, when you reload NZCOM (it will be a cold load, including execution of START), you will not automatically be back in your original directory. End aside.]

Changing Parts of the System

The NZCOM command is not limited to loading whole new operating systems; with a slightly different syntax it can also load individual system modules, rather like the LDR program in a manually installed Z System. There are two important differences, however.

The first is that NZCOM loads code modules (IOP, RCP, and FCP) from REL files rather than from absolute files such as SYS.FCP or DEBUG.RCP. Absolute files can still be loaded using LDR, but this is undesirable under NZCOM, since the addresses of the modules may change as different systems are loaded. NZCOM has the advantage of using a single REL file no matter which system it is being loaded into. In the future, RCPs, FCPs, and IOPs will be distributed in REL form instead of (or in addition to) source code form. The REL file is much smaller and can be used without knowing how to assemble the code.

The second difference is that NZCOM can load command processors and disk operating systems as well. This makes it very easy to change versions of the command processor (with or without security or named directory or submit support, for example) or to experiment with alternative DOSs, such as Z80DOS or P2DOS. This will be a real boon to the development of new operating system components, since one can test new versions so easily and quickly.

For convenience, NZCOM can also load named directory files (of type NDR) and terminal descriptor files (of type Z3T). This is so that you do not have to have LDR.COM on your disk. On an NZCOM system, LDR is a dangerous command, since it does not have safeguards against loading absolute system components to addresses for which they were not assembled. With an NZCOM system, you should remove LDR.COM from your disk.

Other NZCOM Features

There are many more things that could be said about NZCOM that I will save for another time. There is just one more that I want to mention now, and that is the extra "Custom Patch Area" that can be defined with MKNZC (see Figure 1). This option in MKNZC allows one to establish an area in protected memory just below the CBIOS (custom BIOS or real BIOS). This area can be used by various operating system extensions that one wants to preserve from one NZCOM system to another.

Because of the techniques it uses for patching the Z System onto CP/M, NZCOM will not work when a resident system extension (RSX) is present. Thus, for example, you cannot run NZCOM from inside a ZEX script or if DateStamper or BYE is active in low memory (if they are loaded above the CBIOS, there is no problem). I am presently using the patch area for DateStamper. With NZCOM you can effectively have an above-BIOS version of DateStamper without having to move your BIOS down.

I am also planning to experiment with putting BYE in the custom patch area. I think this can be made to work, and it would permit NZCOM to be used on my Z-Node (and I mean used actively — so that the NZCOM system can be changed even from a remote terminal!).

There are special facilities in NZCOM that I do not have the energy to explain now whereby information about a currently running system can be extracted before the new system has been loaded and used to initialize the new system just before NZCOM turns over control to it. This allows an RSX's hooks into the operating system to be maintained.

ZCPR Version 3.4

Now let's turn to the subject of ZCPR version 3.4, which will be released along with NZCOM. Z34, as I will refer to it, is much more an evolutionary step from Z33 than Z33 was from Z30. There are four new features worth pointing out.

Type-4 Programs

The most important and exciting enhancement is the introduction of what is called the type-4 program.

With Z33 I added support for a new kind of program to run under ZCPR3. Programs designed to take advantage of the special features of the Z System have at the beginning of the code a special block of data called the Z3ENV header. This header identifies the program as a ZCPR3 program and contains the address of the ZCPR3 environment descriptor, where all the information necessary to find out about the Z System facilities is available. It also contains a type byte. Conventional Z System programs were of type 1. (Type-2 programs are similar but actually have the entire environment descriptor in the header. Programs of this type are extremely rare. In some senses they are a holdover from ZCPR2 and now obsolete.)

For the new type-3 program I added an additional datum in the Z3ENV header: the starting address for which the code had been assembled or linked. The command processor automatically loads the file to that address before transferring control to it.

Type-3 programs are usually linked to run in high memory (for example, 8000H or 32K decimal) where they do not interfere with most data or code in the TPA. Programs that run as extensions of the operating system (viz. history shells, extended command processors, transient IF processor) or as the equivalents of resident programs (viz. ERA.COM, REN.COM, SAVE.COM) are particularly suitable for implementation as type-3 programs. One cannot always foresee when these programs will be invoked, and it is nice if the contents of memory at the bottom of the TPA are not affected when they do run.

With type-3 programs one must choose in advance the address at which they will run. If the address is too high, there may not be enough room for them to load, and if too low, they are more likely to interfere with valuable TPA contents. In most situations it would clearly be preferable if the program could be loaded automatically as high as possible in memory. I thought of this from the beginning but compromised on the type-3 construct because it was so easy to code.

Joe Wright was not satisfied with this compromise. He soon wrote an initial version of the type-4 program, which does relocate automatically to the top of memory. With a lot of cooperation between us, we have honed it to the point where it functions very nicely and does not add very much code to the command processor.

Because type-3 programs run at a fixed address, albeit not necessarily 100H, they can be assembled and linked in the usual fashion, and the program files contain actual binary object code. Type-4 programs, on the other hand, must be relocatable by the command processor at run time. Thus object code alone is not sufficient.

One possibility would be to use a REL file directly. This would have been very convenient, but the code required to load a REL file is far too complex to include in a command processor running in a 64K memory segment. There is a less familiar relocatable type file known as a PRL (Page ReLocatable) file that, because it restricts the relocation to page boundaries (and other reasons), is much easier to relocate.

A PRL file consists of three parts. The middle part is a standard code image for execution at 100H. After this comes what is called a bit map, where, for each byte in the code image, there is a bit of 0 or 1 to tell whether that byte must be offset for execution at a different page. The bit map is one eighth the length of the code image. Finally, one page (256 bytes) at the beginning of the file serves as a header. This header contains information about the size of the program so that the code that loads it can figure out where the object code ends and the bit map begins.

In the type-4 program, this header is extended to include the code necessary (1) to calculate the highest address in memory at which the program can be loaded and (2) to perform the code relocation to that address using the bit map. The way this is accomplished is somewhat intricate.

The command processor loads the first record of the type-4 file into the temporary buffer at 80H as usual to determine the program type. If it is type 4, the CCP then calls the code in the header. That code calculates the load address and then (this clever idea was Joe's) calls the command processor back to load the program code and bit map into memory at the proper address. When this call is complete and control returns to the header code, it then performs the relocation of the code image at the execution address in memory. Only then is control returned to the command processor for initialization and execution of the program.

SAGE MICROSYSTEMS EAST

Selling & Supporting The Best In 8-Bit Software

- **Plu*Perfect Systems**
  - Backgrounder II: switch between two or three running tasks under CP/M ($75)
  - DateStamper: stamp your CP/M files with creation, modification, and access times ($49)
- **Echelon (Z-System Software)**
  - ZCPR33: full system $49, user guide $15
  - ZCOM: automatically installing full Z-System ($70 basic package, or $119 with all utilities on disk)
  - ZRDOS: enhanced disk operating system, automatic disk logging and backup ($59.50)
  - DSD: the incredible Dynamic Screen Debugger lets you really see programs run ($130)
- **SLR Systems (The Ultimate Assembly Language Tools)**
  - Assemblers: Z80ASM (Z80), SLR180 (HD64180), SLRMAC (8080), and SLR085 (8085)
  - Linker: SLRNK
  - Memory-based versions ($50)
  - Virtual memory versions ($195)
- **NightOwl (Advanced Telecommunications)**
  - MEX-Plus: automated modem operation ($60)
  - Terminal Emulators: VT100, TVI925, DG100 ($30)

Same-day shipping of most products with modem download and support available. Shipping and handling $4 per order. Specify format. Check, VISA, or MasterCard.

Sage Microsystems East
1435 Centre St., Newton, MA 02159
Voice: 617-965-3552 (9:00 a.m. - 11:15 p.m.)
Modem: 617-965-7259 (24 hr., 300/1200/2400 bps, password = DDT, on PC-Pursuit)

The result of this tricky scheme is that most of the type-4 support code that would otherwise have been required in the command processor is in the header instead (this was my contribution to the type-4 concept). Since a PRL file has a two record header anyway (almost all of which is otherwise empty), you get to add this code for free.

Joe pointed out to me some dangers with my type-3 construct. Suppose a type-3 program designed to run at 8000H is somehow loaded to 100H instead. Any attempt to execute it is likely to have less than desirable consequences, to put it mildly. This was not a serious problem with a normal (at the time) ZCPR33 system. Since the command processor would automatically load the type-3 program to the correct address, it took some deliberate action by the user to create the dangerous situation described. Of course, the poor fellow still running ZCPR30 who decided to try out a type-3 program...

However, now that NZCOM is here, the user may very well decide to drop back into CP/M from Z System to perform some tasks. In this situation, a type-3 program is a live weapon, just waiting to blow up the system. The type-4 program poses a similar danger.

We have come up with two defense strategies. One can be implemented in the program itself. There is code (TYP3HDR1.Z80) that can be placed at the beginning of a type-3 program (based on ideas conceived independently by Bob Freed and Joe Wright) that will verify that the code is running at the proper address. This part of the code is, as it must be, address independent (it uses only relative jumps). If the load address is found to be wrong, a warning message is displayed and control is returned to the command processor before any damage can be done. This is the friendlier method, but it makes the programs longer.

The second defense method does not impose any overhead on the program code. It is easier to use than the other method, and it can generally be patched into existing type-3 programs in object form. It can also be applied with type-4 programs, for which the first method cannot be used (type-4 files begin with a relocation header and not with program code, and the system must be prevented from trying to execute the header when the program is invoked under CP/M).

With this method, one places a byte of C7H, the RST 0 instruction opcode, at the beginning of the file. Execution of this instruction causes a call to address 0, which induces a warm boot. This behavior may be puzzling to the user, but at least it does no damage. How, then, will such a program ever execute? The answer is that ZCPR34 checks the first byte of a type-3 program to see if it is a C7H. If it is, the command processor replaces it with a C3H, the JP instruction opcode. To take advantage of this method, the program code must begin with a "JP START" instruction in which the JP is replaced by RST 0 (note: you cannot use JR START instead). The proper assembly language source code is illustrated in Figure 8. Note that the replacement of the RST 0 by a JP is not required with a type-4 program since the header (which is where this construct appears) is never intended to be executed as a standard program, even under Z34.

```
ENTRY:                          ; Beginning of program
        DB      0C7H            ; RST 0 opcode, will become JP
        DW      START
        DB      'Z3ENV'         ; ZCPR3 program ID
        DB      3               ; Type 3
ENVADR: DW      0               ; ENV address filled in by Z34
        DW      ENTRY           ; Execution address
START:                          ; Beginning of main program
```

Figure 8. Form of the Z3ENV header code in a protected type-3 program. An attempt to execute this code under CP/M will result in a warm boot.

The Extended Environment Descriptor and the Drive Vector

The definition of the ZCPR3 environment descriptor has been modified and extended. I will not go into all the details here, but I will describe the main changes.

First, to make some space available for additional important information, the extended ENV eliminates definitions for all but one console and one printer. Eventually there will be a tool (utility program) that allows interactive or command-line redefinition of the characteristics of these single devices so that you will actually have more rather than less flexibility.

The extended ENV will now contain the addresses and sizes in records of the CCP, DOS, and BIOS (actually, the size of the BIOS is not included). This information has been added to deal with problems in some special operating system versions where the CCP and/or DOS do not have their standard sizes of 16 and 28 records respectively, such as in the Echelon Hyperspace DOS for the DT-42 computer. Future versions of NZCOM, which will support variable CCP, DOS, and BIOS modules, will also need this.

Finally, a long needed feature has at last been implemented; a drive vector. The maximum-drive value in the ENV was not adequate in a system with non-contiguous drives (A, B, and F, for example). Now you can tell the system exactly which drives you have on the system, and the command processor will do its best to prevent references to nonexistent drives.

Ever More Sensible Named Directory Security

With Z33 I made it possible to refer by drive/user (DU) to directories beyond the range specified by the maximum drive and maximum user values in the environment provided the directory area had a name with no password. It seemed only reasonable that if a user could access the drive by name, he should be allowed to access it by its equivalent DU as well.

The converse situation, however, was not handled according to similar logic. Suppose the maximum user was 7 but there was a password-protected named directory for user 6. Under Z33 one had the anomalous situation that the user could refer freely to the directory using the DU form but would be pestered for the password if he used the named-directory (DIR) form. This just didn't seem reasonable, and Z34 has corrected this.

Extended ECP Interface

With Z34 I have added an additional option along the lines of BADDUECP. The BADDUECP option allows directory-change commands of the form NAME: or DU: that refer to illegal directories to be passed on to the extended command processor (ECP) instead of directly to the error handler. On my Z-Node, for example, I use the ARUNZ extended command processor to permit references to reasonable facsimiles to the actual directory names to work via alias scripts.

With Z33 attempts to execute a command containing an illegal wildcard character or with an explicit file type would be flagged as errors and passed directly to the error handler. With Z34 one has the option (via the option BADCMDECP) to pass these forms of bad command to the extended command processor as well.

Here are a couple of examples of how this feature can be used with the ARUNZ extended command processor. First, one can enter the following script into the alias definition file ALIAS.CMD:

```
? help $*
```

Now when a user enters the command "?", he will get the help system instead of an error message telling him that he entered a bad command.

You can also use this facility to allow further shorthand commands. With the script definition

```
DIM.Z3T ldr dim.z3t (or nzcom dim.z3t)
```

Now you can load the dim-video TCAP for your system simply by just entering the name of the TCAP file. Using wildcard specifiers in the name of the alias script, you can make any command with a type of Z3T load the corresponding TCAP file. Similarly, entering the name of a library (for example, LBRNAME.LBR) on the command line could automatically invoke VLU on that library. The same concept would allow one to enter the name of a source-code file (for example, THISPROG.Z80 or THATPROG.MAC) to automatically invoke the appropriate assembler (Z80ASM/ZAS or SLR-MAC/M80 for these two examples).

This feature opens another whole dimension for experimentation, and I am sure that users will come up with all kinds of new ways to use it. PLEASE NOTE: if this feature is implemented, you cannot use the old version of ARUNZ that I so painstakingly documented in my last column (alas, barely born and already obsolete). Previous versions of ARUNZ used '?' and '.' for special purposes. Those characters were carefully chosen because they could never appear in command names passed to ARUNZ, but now they can! Therefore, in version 0.9H of ARUNZ I have changed these characters to '_' (underscore) instead of '?' and ',' (comma) instead of '.'.

That's it for this issue, I'm afraid. I still didn't get to a discussion of defects in the shell coding for WordStar 4 (I hope these will be corrected in version 5, which is apparently really in the works at this time). My discussion of the ZEX in-memory batch processor and the improvements I have been making to it will also have to wait still longer. ■

Language and Compiler Design News

LALR, an LALR(1) parser generator, runs on MS-DOS computers and generates automata-based parsing engines for computer languages, command languages, and macro languages.

LALR reads a BNF grammar specification and outputs source code in Microsoft C, Turbo C, and UNIX C. It has successfully handled a COBOL grammar of 1561 productions, generating a parser with 2187 states in about 60 seconds.

LALR generated parsers are capable of processing the target language at a rate of approximately 14,000 lines per minute on an IBM PC/AT. Time includes scanning and parsing.

LALR provides an easy way to interface with a lexical scanner and the rest of your system. An advanced error recovery technique provides good repairs for syntax errors.

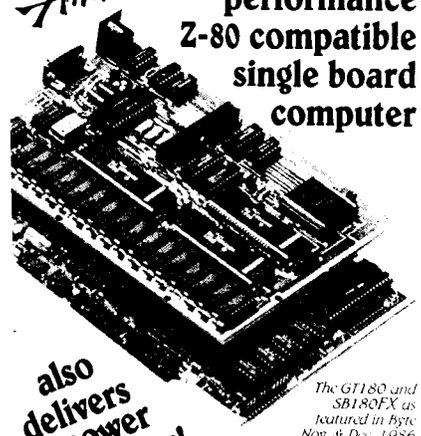
LALR comes with grammars for Ada, BASIC, C, and Pascal, source code for a lexical scanner, parser skeleton, test program, and calculator. It has a 60-day money-back guarantee. Price is **$99.**

LALR Research

714-832-LALR

1892 Burnt Mill / Tustin CA 92680

Announcing **A high performance Z-80 compatible single board computer**



The GT180 and SB180FX as featured in Byte Nov. & Dec. 1986

also delivers power graphics!

The SB180 FX

- *Small, fast, memory-packed single board computer*
- *add the Micromint GT180 for high resolution graphics*

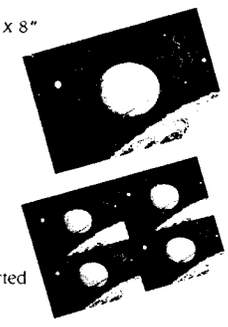
SB180 FX features:

- Measures only 5.75" x 8"
- 64180 CPU running at 6, 9 or 12 Mhz
- Up to 512K bytes RAM and 32K bytes ROM
- Two 38.4 baud serial ports
- A parallel printer port
- Peripheral expansion bus
- Three bi-directional parallel ports
- Industry standard 765A - compatible disk controller
- NCR 53C80-SCSI bus controller for hard disk or network communications

■

GT180 features:

- Measures only 5.75" x 8"
- Designed to piggy-back on top of the SB180 or SB180 FX
- High resolution of 640x480x16 colors from a palette 4096
- Advanced HD63484 CRT controller
- 38 commands including 23 graphic drawing commands
- Fully software supported by Borland's GT180 Graphix Toolbox and Modula-2
- 2 million pixels per second



■

SB180 FX .....as low as $409.00
GT180 .....as low as $395.00

Turbo Modula-2.....$69.00
Turbo Modula-2 w/GT180 Graphix Toolbox.....$89.00

To order call

1-800-635-3355

TELEX: 643331

For technical information call 1-(203)-871-6170

MICROMINT, INC.
4 Park St., Vernon, CT 06066

Designing Operating Systems

A ROM Based O.S. for the Z81

by Clark Calkins, C.C. Software

Reading the article "Double Density Floppy Disk Controller" in TCJ #30 brought back fond memories of my CP/M days. The operating system was designed as a minimal system allowing the user to make the most out of his hardware. As the article pointed out, system performance was really tied to the design of the BIOS. A good design resulted in a fast system and a poor (or conservative) design yielded a slow system. While most users did not want to rewrite their BIOS code, it was certainly within the reach of the average assembly programmer.

One project I started a number of years ago (and haven't finished yet) was to build a small and portable computer that I could use to monitor real world functions. A data collector and with data reduction capabilities if you will. I wanted a system that could monitor outside signals, record them, do minimal processing and possibly display some results. I had visions of an automotive diagnostic system that could do all kinds of wonderful things. Have you ever tried to find a squeak in a car? It only happens when you are driving and then you can't seem to pin point the location of the noise. It would be nice if I could position a few microphones around the inside of the car and record the sounds as I drove over a bumpy road. With the data I could later isolate the offending signals and locate the source by triangulation. A bit involved, but it COULD work. Or how about recording and analyzing the spark plug voltage. You can really find out about the condition of the engine by looking at the voltage transients. There may be some applications around the house that this could be used for. For any of these projects, suitable analog to digital converter interfaces would be required. There are some inexpensive A/D chips (8-bit) that run pretty fast (10k to 100k samples per second). It wouldn't be too difficult to build a board with four or eight channels. I would get to these details when the time comes (famous last words).

Now at the time I was thinking of all this I was involved with Exatron (a small Silicon Valley Company) as a Saturday afternoon "handyman" programmer. This company, among other things, was producing a very small digital tape drive (called the Stringy Floppy) that was interfaced to various systems as an inexpensive storage device to replace cassette recorders. First it was for S-100 computers, then Radio Shack TRS80s and Apples, and finally the Timex/Sinclair ZXs. The tapes used were small (3" × 1.25" × 0.125") endless loop cartridges that could hold 10k to 100k bytes depending on the length of tape (5 to 50 foot tapes were available). I had a prototype of the Timex interface to play with and this was really neat except that all you could do was read and write BASIC source files. What I wanted was a minimal operating system that used the tape as a "slow disk drive." After all, the Timex had 16k of RAM, a Z-80 processor and it would easily run off of a twelve volt car battery (less monitor), so why not?. Just about this time Timex was getting out of computer business and the systems were being sold at drug stores (even Safeway) for almost nothing. I was really intrigued by this and I set out to write an operating system for the little machine.

For those who don't know or cannot recall, the ZX-81 had a BASIC in ROM addressed 0-8k and a 16k RAM package addressed from 16-32k. Third party vendors were selling 32k RAM packs for the remaining address space. The hardware design prevents execution of Z-80 instructions beyond the 32k address but data could be stored there. To make matters worse, the video screen was memory mapped in RAM and the system used its own character set (not ASCII). Anyway, my plan was to write an operating system that looked like CP/M with a BIOS that managed the screen and tape I/O. This would be burned into an EPROM and sit in the 8-16k address space. I would use the basic for data reduction (it was too slow for data collection) so it had to be able to read and write data files from the tape.

I chose to emulate the CP/M operating system because it was small, simple, and I had the source code for it (via my Source Code Generators, see reference 1). I had written many different BIOS's in the past so I knew I could do one more. For anyone that has had to type on the ZX-81's membrane keyboard, you know what a pain this was. I wanted to simplify typing as much as possible. So I would use "function keys" (using the numeric keys) to run all of the built in commands. I realized that I would have to write any programs that I wanted to run but I thought these would be short assembly data collection programs. I didn't plan to run WordStar or anything like that.

I used another computer system (my trusty Digital Group) to do the software development and then I would down-load the ZX-81 via the cassette interface. A slow way to send data, but it was simple to implement and it worked. This way I could take advantage of the video editors and assemblers I had until I got the ZX-81 running. A small assembly routine running on the ZX-81 (using POKE's from BASIC) would read the test program into RAM and jump to it.

Now CP/M was written for an Intel 8080 cpu to be compatible with the early home computer systems. Because I was really limited in memory space, I wanted to convert it to Z-80 code. Changing to relative jump instructions alone would save ½k. In doing this, I cleaned up the code a lot, rearranged the routines (trying to shorten it by eliminating jump instructions), and removed all of the needless code. I managed to squeeze about 1.5k from its normal length but the program was still basically CP/M. Now the fun came. Make CP/M talk to the tape drives in a reasonably efficient manner. CP/M normally thinks it's talking to a rotating disk drive and a fairly fast one at that. Now I had slow tape drives to deal with. The standard routine to allocate free disk blocks tried to pick new blocks from the empty pool that were as close together as possible without regard to the direction of rotation. Not too bad for a floppy disk, but for a tape drive this was ridiculous. I had to first look in the forward direction (remember the tapes are an endless loop) for empty blocks and as a last resort look backwards. In fact even a floppy or hard disk drive could benefit from this logic. Other changes were made to the BDOS, like automatic disk re-logging (reference 2) and file searching over multiple volumes. I wanted to see how well I could make the system perform. Now CP/M accesses the directory a lot. To make the access time reasonable, I decided to keep the directory in a separate buffer that only had to be read once, and

write out only the changed sections. Control-C (and warm boot) processing was modified. Since the operating system will be in EPROM, only the disk parameters have to be reset. To help in the writing of other programs (assemblers editors), extra BDOS entry points were included for file opening, character read and write, and direct block reads and writes.

While I was at it, I might as well change the command processor interface (CCP). I wanted as many built in commands as I could fit into the limited memory set aside for the operating system. I replaced PIP with a COPY command (with query on wild card names) and inserted a limited STAT command. I included a DDT like debugger but it does lack the disassembler portion. Additionally, a SUBMIT facility was included that automatically recognized a command file with the SUB extension rather than COM. When a command would be entered without an extension, a search would be made for "filename.COM" and "filename.SUB". If only one of these was found, it would be executed. Otherwise an error results.

The poor keyboard (mentioned previously) guided me into this "revelation." If the first character typed in a command line was a digit, then I would substitute one of the built in commands. Similar to function keys on other machines. Thus when "2" is typed, "DIR" pops up ready for a drive or filename. An added benefit is that transient commands are any commands not beginning with a digit. I no longer had to search a built in command table. This feature worked so well, I immediately incorporated it into my other CP/M computers.

When the project had progressed to this point, I was ready to test the software on the ZX-81. I would load the operating system into RAM for testing until I found a way to burn an EPROM. The first time I loaded the program and ran it... CRASH! In fact for the first few days, every time the program was run it crashed and the ZX-81 wasn't being helpful in telling me why. After much bug hunting I got the software to run a few seconds before crashing! At least the video translations were working (ZX-81 to/from ASCII). I could see the sign-on message but it would crash shortly there after. I decided to test the tape routines separately just to see what portion of the program could not function. I incorporated the tape I/O routines into a rudimentary format program (I needed one of these anyway of course). This lacked any screen output. It just waited for an key (any key) and formatted the tape. If successful, it waited a few seconds and tried to read it back. There were surprisingly few bugs here considering that the tape was COMPLETELY software driven. Now back to CP/M. Using the built it DDT and setting break points all over the place, I traced the problems to the initializing routines. The code seemed simple, but executing it would crash the system. When the ZX-81 comes up in BASIC, the screen memory is in the lowest portion of the 16k of RAM. I wanted to move this to the upper portion, store the CP/M parameters next, and place the stack just below this. Then all RAM from 16k to the bottom of the stack would be available for a program to run. The screen driver is interrupt driven and hardwired into the BASIC and it is VERY touchy. After a few weeks of frustration, I found out that the stack had to be above the screen. Just a quirk of the interrupt routines. With this straightened out, the system came up and I was in business! There was the usual debugging the new code for the commands but this was straight forward and I didn't mind; it was expected.

Now I finally had a system running that could save memory to tape, get directories, load programs (if there were any to load), etc. I felt like I had really accomplished something! But before I could USE the system, I had to have some facilities to write programs. To do this I wrote an assembler (8080 code at first, Z-80 would come later) and a simple line editor. Using this I fixed up the format program as a test case. I added messages and error recovery etc. Typing on the ZX-81 is very cumbersome, but at least I could develop programs in a CP/M environment. Who else could say that? Things certainly weren't done yet, but at least the system was working.

Now I can sit back and relax for a while. This has been a very long project (about six months from when I started) and other things where getting my interest. After all I have bills to pay and this certainly wasn't going to help.

My original intent was to come up with a portable data collection computer that I could use for various other projects. There is still work to be done before I will be able to do this. I have to burn an EPROM (presently I am limited to 8k of RAM for program storage as the operating system takes up the other 8k). An analog to digital converter interface needs to be built. I will probably wire wrap an ADC8004 (8-bit A/D) as a first step. Some amplifiers will certainly be needed for this as the signal levels will need to be boosted to the 0-5 volt range.

I hope to get back to this shortly. After all I have 6000-8000 lines of code and many months invested in a $29.95 computer. But it has been fun and that's what counts.

Clark A. Calkins
C.C. Software
1907 Alvarado Ave.
Walnut Creek, CA 94596

References

1) SCG22, A Source Code Generator for CP/M 2.2, C.C. Software, 1907 Alvarado Ave, Walnut Creek, CA 94596.
2) *Automatic Disk Re-Logging With CP/M 2.2*, Clark A. Calkins, Micro Cornucopia, April-May 1985 (issue #23).

IS NOTHING SACRED?

Now the FULL **source code** for **TURBO Pascal** is available for the **IBM-PC!** WHAT, you are still trying to debug without source code? But why? Source Code Generators (SCG's) provide completely commented and labeled ASCII source files which can be edited and assembled and UNDERSTOOD!

SCG's are available for the following products:

__ TURBO Pascal (IBM-PC)* ..... $ 67.50
__ TURBO Pascal (Z-80)* ..... $ 45.00
__ CP/M 2.2 ..... $ 45.00
__ CP/M 3 ..... $ 75.00
* A fast assembler is included free!

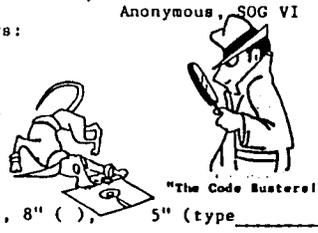
"The darndest thing I ever did see..." Pournelle, BYTE

"I have seen the original source and yours is much better!" Anonymous, SOG VI

The following are general purpose disassemblers:

__ Masterful Disassembler (Z-80) .. $ 45.00
__ UNREL (relocatable files) (8080) $ 45.00

VISA/MC/check card# ____ ____ ____ ____ expires __/__
Shipping/Handling $ 1.50
Tax $_____
Total $_____



All products are fully guaranteed. Disk format, 8" ( ), 5" (type_____).

C.C. SOFTWARE, 1907 ALVARADO AVE., WALNUT CREEK, CA. 94596, (415) 939-8153

CP/M and TURBO Pascal are trademarks of Digital Research & Borland Int.

Advanced CP/M

Boosting Performance

by Bridger Mitchell

CP/M — an anachronism for many, a cuss word nearly forgotten, and a bewitching mistress who dwells in mystery near the heart of our hobby. This new column will court her fancy, defer to her eccentricities, and, yes, expose her vulnerabilities. I'm writing it to share a few tricks and concepts I've learned, recycling to the CP/M community a little of what I've received from them.

CP/M® — the letters abbreviate Gary Kildall's original name *Computer Program/Monitor* — has evolved into a number of important strains and variations. Digital Research's mainstream release, which probably remains the most widely used version of CP/M today, is CP/M 2.2. DRI's CP/M 3.1 (aka CP/M Plus) is also actively used. More recently, Z80 "clones" of the operating system have gained increasing favor, including ZRDOS from Echelon, QP/M from MicroCODE,, and recently P2DOS and its descendants PZDOS and Z80DOS. Then there are the networking variants — MPM® from DRI and TurboDOS® .

What is CP/M, today? For our purposes, it's some version of this venerable operating system, the granddaddy of portable OSs for microcomputers. Equally important, it's a diverse group of users who share a sense of community, swap ideas and advance each other's innovations. Many of us who still belong to that community continue to use CP/M actively, not least because we find this view of computing constructive and personally rewarding. Bulletin boards, joint projects, user groups and publications like TCJ keep us in touch.

Advanced CP/M will dip into various CP/M and Z80 topics, emphasizing advanced programming techniques. Our watchwords will be: portability, compatibility, performance, and operating system principles. I'll try to serve up a mixed diet — OS concepts, hints and tricks, occasional advice, and a healthy dose of usable code extracts. Topics I hope to cover in the next columns include: what makes it tick (or, Why did the input character disappear until I exited the program?), and how to teach the old DOS new functions.

This first column's subject is performance — ways to boost the CP/M octane rating. We investigate two quite different areas — logging in new disks quickly, and the gains from hand-optimizing Z80 code.

Fast Disk Resets

A fundamental operating system activity is to manage storage. The CP/M BDOS keeps a *free list* of available data blocks for each disk. Writing a file allocates one or more blocks; deleting a file releases its blocks. CP/M keeps the list as a *bitmap*, with a bit set for each block that has been allocated. In another column we'll delve into the fine points of CP/M's management of the free list. Our concern now is to ensure that the list is correctly up to date.

Bridger Mitchell is a co-founder of Plu*Perfect Systems. He's the author of the widely used DateStamper (an automatic, portable file time stamping system for CP/M 2.2); Backgrounder (for Kaypros); BackGrounder ii, a windowing task-switching system for Z80 CP/M 2.2 systems; JetFind, a high-speed string-search utility; and most recently DosDisk, an MS-DOS disk emulator that iets CP/M systems use PC disks without file copying.

Bridger can be reached at Plu*Perfect Systems, 410 23rd St., Santa Monica CA 90402, and via Z-Node #2, (213)-670-9465.

LISTING 1

```
; Routine:      FRESET -- fast drive reset and login
; Author:       Bridger Mitchell (Plu*Perfect Systems)
; CPU:          Z80-compatible
; Date:         January 2, 1988
; Version:      1.0
;
xbdos   equ     5

;       Fast-reset drive (A)
;       A = 0 ... 15  for  A: ... P:
;
CSEG
freset:
        ld      (reqdrv),a      ; save requested drive
        push    af
        ld      c,24            ; get vector of logged-in drives
        call    xbdos
        pop     af              ; (recover requested drive)
        push    hl              ; save logged-in vector
        call    fshftr          ; shift requested drive's bit to bit 0
        bit     0,l             ; is requested drive logged in?
        pop     hl              ;  (recover (unshifted) logged vector)
        jr      z,flogit        ; ..z - no, just log it in
        ld      e,0             ;  initialize drive index/count
        jr      frese2
;
fresel: ld      a,l             ; shift vector right 1 bit
        call    fshftr
frese2: bit     0,l             ; if drive is not logged in
        jr      z,frese3        ; ..check next drive
        ld      a,(reqdrv)      ; or if it is the requested drive
        cp      e               ;
        jr      z,frese3        ; ..check next drive
;
; found a second logged-in drive, so switch to it
;
        call    fslctit         ; select e'th drive
        call    flogout         ; log out requested drive
        jr      flogit          ; then log into it and exit.
;
frese3: inc     e               ; increment drive count
        ld      a,e
        cp      16
        jr      c,fresel        ; .. and continue for 16 drives
;
; no second drive found,
;
        call    flogout         ; log out requested drive
        ld      c,13            ; do general reset, then (re)log requested
        call    xbdos           ;
```

The free list must be constructed, from allocated block numbers stored in directory entries, the first time a disk is used. And to preserve the integrity of the file structure, the list must be reconstructed when a (floppy) disk is removed and another inserted in the same drive. It must also be reconstructed whenever any disk is modified by writing to it without using the BDOS.

CP/M provides two functions that will (eventually) cause a new free list to be constructed — Reset All Drives (function 13) and Logoff Drive(s) (function 37). If you've ever used the original CP/M 2.2 BDOS you've almost certainly experienced that sinking feeling that comes after changing floppy disks without resetting the drive and abruptly seeing the dreaded "BDOS Error on x: Read Only" message, followed by immediate termination of the program and loss of all your work.

To prevent this disaster, most programs use function 13 whenever prompting for a disk change. This works, but it's a monkey wrench that requires CP/M to rebuild the free list for every drive, not simply the one with the new disk.

```
;
; log into requested drive
;
flogit: ld      a,(reqdrv)
        ld      e,a
;
fslctit:ld      c,14            ; select bdos drive
jxbdos: jp      xbdos
;
; log out the requested drive
;
flogout:ld      a,(reqdrv)      ; set up bit to log out drive
        ld      hl,1
        call    fshftl
        ex      de,hl
        ld      c,37            ; log out drive in DE vector
        jr      jxbdos
;
;
; shift hl right (a) bits
;
fshftr:
        inc     a
shftr1: dec     a
        ret     z
        srl     h
        rr      l
        jr      shftr1
;
;
; shift hl left (a) bits
;
fshftl:
        inc     a
shftl1: dec     a
        ret     z
        add     hl,hl
        jr      shftl1

;--------------------
DSEG
reqdrv: ds      1

        end
```

LISTING 2

```
(*  Sieve of Erasthones Benchmark (Jan. '83 BYTE) *)
(*  Turbo Modula-2 *)
(*  Compile with test, overflow switches off and native code on. *)

MODULE prime;

CONST
  size = 8190;

VAR
  flags : ARRAY [ 0 .. size ] OF BOOLEAN;
  i, prime, k, count, iter : CARDINAL;
  ch : CHAR;

BEGIN
  WRITELN("Type Return"); READLN(ch); (*  WRITELN("10 iterations"); *)
  FOR iter := 1 TO 10 DO
    count := 0;
    FOR i := 0 TO size DO flags[i] := TRUE END;
    FOR i := 0 TO size DO
      IF flags[i] THEN
        prime := i + i + 3;
        k := i + prime;
        WHILE k <= size DO
          flags[k] := FALSE;
          k := k + prime;
        END;
        count := count + 1;
      END;
    END;
  END;
  WRITELN( count, " primes" );
END prime.
```

CP/M provides function 37 for rapid, pinpoint disk resets, but it has acquired a fearsome reputation and is almost never used, because Digital Research coded this section of CP/M 2.2 with a fatal bug. The precise conditions are complex, but if you use Function 37 to reset the *current* drive and then write to that drive before logging in any other drive, CP/M 2.2 will write the file to the wrong data groups, very likely destroying some existing files.

Nevertheless, function 37 can indeed provide safe, fast resets. The trick, which was described by a CP/M veteran years ago (whose name I have now lost), is to log into a *different* drive before calling the function. The general-purpose FRESET routine (Listing 1) does just that. Think of it as a precision socket wrench; a tool designed expressly for logging in a new disk. You call it with the drive to be reset in register A; it resets the drive and then logs it in.

This is how it works: Because a program will not normally know what drives are available on the host system, the routine uses Function 24 to get the 16-bit vector of currently logged-in drives. (A *logged-in* drive is one for which CP/M has already constructed a free list). If the requested drive is not already logged in, all that needs to be done is to have CP/M log it in; CP/M will build its free list as part of the login operation.

More commonly, the requested drive will be active; that's the one the program wants to use for a new disk. So FRESET next checks for a second logged-in drive. If there is one, it logs it in, and then it resets the requested drive.

At this point, if a second drive has been logged in, CP/M 2.2 will be happy and we can proceed to login the requested drive and exit. But, if there was no second drive, the requested drive is still, in some sense, the active drive, even though it was "logged off." Therefore, FRESET must use the general reset Function 13 to reset "all" drives, even though only the one drive is logged in.

In pseudo-code, the FRESET algorithm is:

```
if requested_drive is logged in
    if a second drive is logged in
        log into the second drive
    logoff requested drive (fn.37)
        if no second drive was logged in
            do general reset (fn. 13)
log into requested drive
```

FRESET is the natural routine to use in a file-copy program. For example, I used it in DATSWEEP (the DateStamper file maintenance utility) to re-log the destination disk before each single or group copy, allowing the user to change disks between each copy operation without further commands.

FRESET is also the right routine to use after running any program that directly

modifies a disk directory, such as UNERASE or DU. That type of utility modifies the file structure "behind the back" of the BDOS. When that happens it is essential that the BDOS completely rebuild the disk's free list to reflect the changes. Under CP/M 2.2 it would be safe, but slow, to use Function 13 for this. However, recent version of ZRDOS and PZDOS handle the free list differently for non-removable (hard and ram) drives. They do this to speed up warm-boots and general resets by skipping the reconstruction of the free list for those drives. Therefore, with these DOSs the program *must* use Function 37 to get the free list rebuilt.

It's highly desirable for fundamental routines to be compatible with as wide a range of systems as possible. If they are, they can be dropped into application programs and trusted to work without special checking. FRESET passes the portability requirement; it should serve any flavor of CP/M that is running on a Z80-compatible processor.

A Very Hot Sieve

Most TCJ readers have probably encountered comparative timings from the Sieve of Eratosthenes routine, introduced in the January, 1983 *Byte*. The sieve counts the first N prime numbers, and it has been widely used as a (too) simple benchmark for different compilers and languages.

Just before Echelon released Turbo Modula-2, I received a copy of this new compiler from Frank Gaude' and the test file in Listing 2. Frank's comparisons indicated that TM2 was a hot compiler, bettering the performance of Turbo Pascal on both Z80 and 8088 machines, and a variety of other Pascal, Modula-2, and C compilers. Those figures set me to wondering — as a code generator, how close does TM2 come to the theoretical maximum performance? Over the lunch hour I used a napkin to sketch out the first version of a Z80 assembly language sieve. That evening's testing showed it was competitive, and so I decided to see if I could push the Z80 to its limit. JETPRIME was the result. Listing 3 contains the code, with the number of tstates required for each instruction as calculated by the SLR+® assembler for the HD64180® cpu.

Optimization requires mental iteration. My first sketch was to mimic what a compiler might do — translate each line of high-level language code as it stood and use memory locations for all variables. I knew that pedestrian approach would run, but of course it would really only be walking.

My first improvement cycle focused on using the cpu's registers efficiently, substituting register moves for load/store

Listing 3

```
                        ; JetPrime.ASM
                        ;
                        ; Z80 Sieve of Eratosthenes Benchmark.
                        ; Bridger Mitchell, Piu*Perfect Systems 2/29/88
                        ;
                        ; See Jan. '83 BYTE
                        ;
                        ;
        0001            ABSOLUTE equ 1                  ; set to 0 for relocatable routine
        1FFE            SIZE    equ     8190            ;largest integer to be checked
        076B            ANSWER  equ     1899            ;number of primes in [0...SIZE]
                        ;
        0005            BDOS    equ     5
        000D            CR      equ     0dh
        000A            LF      equ     0ah
                        ;
        000A            NITER   equ     10              ;number of iterations, for timing.
                        ;
        0001            IF ABSOLUTE
                        ;
                        ; last memory address used is:
        2200            top     equ     free + (SIZE + (100H - (00FFH AND SIZE)))
                        ;
        0022            pagend  equ     (top shr 8) and 00FFH   ;page following flag[SIZE]
                        ;
        0201            flags   equ     top - (SIZE +1)         ;array of boolean flags
                                                                        ;total of 0...SIZE = SIZE+1
        0100                    org     100h
                        ENDIF


addr.   op code  HD64180
                 tstates

 0100   C3 013F      9          jp      test
                        ;
                        ;
                        ; The JETPRIME SIEVE ROUTINE
                        ;
 0103                   primes:
        0104            f1 equ $+1
 0103   21 0201       9          ld      hl,flags        ;set all flags TRUE
        0107            fplus1 equ $+1
 0106   11 0202       9          ld      de,flags+1
 0109   01 1FFE       9          ld      bc,SIZE         ;count = total - 1
 010C   36 01         9          ld      (hl),01h
 010E   ED B0    14*BC-2         ldir
                        ;
                        ; Initialize the registers.  They will be used as:
                        ;
                        ; A  : termination condition
                        ; BC : count of primes found
                        ; DE : i = index
                        ; HL : pointer to flags[i] element
                        ; BC': base of flags array
                        ; DE', HL': working registers for inner loop
                        ;
                                                        ;bc = count = 0 already
 0110   50           4          ld      d,b             ;de = i = 0
 0111   58           4          ld      e,b
        0113            f2 equ $+1
 0112   21 0201       9          ld      hl,flags        ;&flags[0]
 0115   D9           3          exx                     ;bc' = &flags[0]
        0117            f3 equ $+1
 0116   01 0201       9          ld      bc,flags
 0119   D9           3          exx
        011B            p1 equ $+1
 011A   3E 21         6          ld      a,pagend-1      ;load a = last page of flags[]
                        ;
 011C   CB 46         9  loop:   bit     0,(hl)          ;if flag[i] == TRUE
 011E   CA 0138 06/09            jp      z,nexti
                        ;
 0121   D5           11 innerip:push    de              ;..set registers for inner loop
 0122   D9           3          exx                     ;..and count this prime
 0123   E1           9          pop     hl              ;hl' = i
 0124   54           4          ld      d,h             ;de' = i
 0125   5D           4          ld      e,l

 0126   29           7          add     hl,hl           ;*2
 0127   23           4          inc     hl
 0128   23           4          inc     hl
 0129   23           4          inc     hl              ;+3
 012A   EB           3          ex      de,hl           ;de' = prime = i + i + 3, hl' = i
 012B   19           7          add     hl,de           ;hl' = k = prime + i
 012C   09           7          add     hl,bc           ;hl' = &flag[k]
                        ;
 012D   BC           4  whil:   cp      h               ;while k <= SIZE
 012E   38 06    06/08          jr      c,whilx
 0130   CB 86       13          res     0,(hl)          ;..flag[k] = FALSE
 0132   19           7          add     hl,de           ;..hl' = k = hl' + prime
 0133   C3 012D      9          jp      whil
                        ;
 0136   D9           3  whilx:  exx                     ;count the prime
 0137   03           4          inc     bc              ;count++
                        ;
                        ; do next i
                        ;
 0138   23           4  nexti:  inc     hl              ;&flags[i]++
 0139   13           4          inc     de              ;i++
 013A   BC           4          cp      h               ;while i < size
 013B   D2 011C 06/09            jp      nc,loop         ;..loop
 013E   C9           9          ret                     ;done, bc = count of primes
```

```
                        ;
                        ;
                        ; The re-executable JETPRIME DRIVER CODE.  Uses the CCP stack.
                        ;
013F                    test:
013F  CD 01E0     16+   calinit:call    init            ;do any initialization
0142  11 0183      9            ld      de,signon       ;banner & prompt
0145  0E 09        6            ld      c,9
0147  CD 0005     16+           call    bdos
014A  CD 0172     16+           call    waitcr          ;wait for CR to start up
                        ;
014D  3E 0A        6            ld      a,NITER         ;set # of iterations
014F  F5          11    more:   push    af              ;repeat the prime
0150  CD 0103     16+           call    primes          ;  calculation
0153  F1           9            pop     af              ;    several times
0154  3D           4            dec     a               ;      to improve timing
0155  C2 014F 06/09              jp      nz,more         ;        resolution.
                        ;
0158  21 076B      9            ld      hl,answer       ;verify answer
015B  ED 42       10            sbc     hl,bc           ;(cy is clear)
015D  11 01C9      9            ld      de,donemsg      ;if Z,
0160  28 03    06/08            jr      z,sayend        ;..say it's correct
0162  11 01D7      9            ld      de,badmsg       ;else say it's bad
0165  0E 09        6    sayend: ld      c,9
0167  CD 0005     16+           call    bdos
                        ;
016A  11 01B9      9    exit:   ld      de,retmsg       ;ask for CR again
016D  0E 09        6            ld      c,9             ;..and exit to CCP
016F  CD 0005     16+           call    bdos
                        ;
0172  0E 06        6    waitcr: ld      c,6             ;wait for a CR
0174  1E FF        6            ld      e,0ffh
0176  CD 0005     16+           call    bdos
0179  FE 0D        6            cp      CR
017B  20 F5    06/08            jr      nz,waitcr
017D  5F           4            ld      e,a             ;echo a CR
017E  0E 02        6            ld      c,2
0180  C3 0005      9           jp      bdos
                        ;
0183  0D 0A 4A 65       signon: db      cr,lf,'JetPrime Z80 '
0192  2D 2D 20 42               db      '-- BYTE Sieve Benchmark - 10 iterations'
                        ;
01B9  0D 0A 48 69       retmsg: db      cr,lf,'Hit <RETURN>.$'
01C9                    ;
01C9  0D 0A 31 38       donemsg:db      cr,lf,'1899 Primes$'    ; the correct answer
01D7                    ;
01D7  0D 0A 57 72       badmsg: db      cr,lf,'Wrong!$'
01E0                    ;
                        ;
        0001            IF ABSOLUTE
                        ;
01E0  C9            9   init:   ret                     ; dummy routine in absolute version
                        ;

01E1    001F                    ds      [100h - [$ AND 00ffh]]  ;filler
                        ENDIF
                        ;
0200    0000            free:   ds      0               ;data area, page-aligned
0200                    ;
        0000            IF ABSOLUTE eq 0
                        ;
                        ; INITIALIZATION CODE for RELOCATABLE VERSION
                        ;
                        init:
                                xor     a               ; prevent re-entry
                                ld      hl,calinit      ;   of this code
                                ld      (hl),a          ;     by storing 3 nop's
                                inc     hl              ;       at call location
                                ld      (hl),a
                                inc     hl
                                ld      (hl),a
                        ;
                                ld      hl,free         ; calc flags[0] address
                                ld      de,SIZE+1
                                add     hl,de           ; hl = min. addr. for flags[SIZE+1]
                                ld      a,l             ; if not on page boundary
                                or      a
                                jr      z,init1
                                ld      l,0             ; .. adjust to next page
                                inc     h
                        init1:  ld      a,h
                                dec     a
                                ld      (p1),a          ; install pagend address
                                sbc     hl,de           ; deduct sizeof flags[]
                                ld      (f1),hl         ; install flags addresses
                                ld      (f2),hl
                                ld      (f3),hl
                                inc     hl
                                ld      (f1plus1),hl    ; install flags+1 address
                                ret

                        flags   equ     100h            ; dummy values
                        pagend  equ     1h              ; "     "
                        maxtop  equ     free + (SIZE+1) + 0ffh  ;worst-case size of routine
                                ENDIF

                                END
```

from memory, and using the stack when there were no idle registers (push + pop is 20 tstates, store + load is 32-40). A smart compiler, using *register variable* declarations, might be able to do almost as well, and the Borland implementation of Modula-2 does provide for four 16-bit register variables.

The Z80 has a fair number of registers, so the trick is balancing off how to use them to best advantage. It usually takes several tries of various combinations, to get things really tight. I gained further efficiency by pre-loading key constants into registers outside the inner loops (lines 112 and 116).

With most of the code now streamlined, the condition-testing at the end of both loops stood out as both slow and cumbersome. Suddenly I recognized that if I could cause the address of the end of the array to fall at the beginning of a 256-byte memory page, the test for completing the loop could be done by comparing the high byte of the array pointer with a limit value. By arranging storage this way I could preload the accumulator with the final page value (line 11A) and use it for both loop-termination tests (lines 12D and 13A). Actually, it's rather unusual to have a double loop in which the accumulator never changes!

The final JETPRIME is a screamer, some 140 percent faster than the TM2 code (4.1 seconds vs. 10, 4Mhz Ampro Little Board)! Is it the ultimate Z80 sieve? Possibly, but I imagine some TCJ reader will squeeze out a few more tstates in his lunch hour musings. (Send your race horses in; we'll publish the new winner!)

What's the point of a faster sieve? Well, I wrote JETPRIME primarily because the performance mountain was there to be climbed. But there's also a practical side. Some applications have very high-frequency tight loops, and 100 percent or better improvements are well worth achieving by hand-coding, provided the assembled code can be smoothly integrated with the high-level language. JETPRIME stands as a small example of what might be accomplished

Listing 3 was assembled with the ABSOLUTE flag set, to set up the page-aligned addresses. However, some simple initialization code (line 200) will convert it into a fully relocatable routine; the code is generated when ABSOLUTE is 0.

The one-time initialization code modifies instructions in its own code segment, first to change the instruction that calls it (at 13F) into 3 nop's, so that it won't be re-executed, and then to calculate page-aligned addresses and set the address pointers in the code. (The primes routine might get re-executed by a GO command to ZCPR, or if imbedded in a larger program). I've caused the initialization code to share memory with

the buffer area, a useful design that can be used to reduce memory requirements when you have one-time startup code. This is pointless for a stand-alone sieve on a 64K machine, of course, but is worth keeping in mind for any routine integrated into a larger program.

"Self-modifying" code is pejorative in some programmers' lexicons. My personal dictionary includes the advisory: know its limitations, and use it to advantage. Self-modifying routines are harder to debug and require special attention to re-execute correctly without side effects. And they may not run properly on processors with pipeline architectures, separate code and data segments, or associative (cache) memories. In particular, JETPRIME might require revision to run successfully on the current-mask version of the Z280. Of course, for that chip further optimization may be possible also. (If you are able to test this one, drop us a line!).

A note on timings. The real-time performance of JETPRIME, or any program, will vary with cpu clock speed, memory access speed (wait states), and interrupt processing. A cleaner test would disable interrupts to avoid some timing differences between systems with the same clock speeds. (In an actual application it may be necessary to leave interrupts running, or perhaps to mask only low priority interrupts.) Timing comparisons should omit all but minimal input/output, to avoid spurious differences due to variations in message routines and limited I/O channel bandwidth; the TM2 code incurs some overhead in formatting the value of 'count'.

Although JETPRIME is of interest in its own right, its real mission is to remind us that hand-optimized code can, at key points, significantly boost the performance of CP/M systems.

Next Time

I hope many of you readers of *Advanced CP/M* will talk back, suggest topics of interest, contribute better mousetraps, and pounce on your columnist's mistakes. So send postcards!

In the next column: extending CP/M while avoiding fratricide in the brave new RSX world. ■

MOVING?

Make certain that TCJ follows you to your new address. Send both old and new address along with your expiration number that appears on your mailing label to:

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, MT 59912

If you move and don't notify us, TCJ is not responsible for copies you miss. Please allow six weeks notice. Thanks.

• Z Best Sellers •

Z80 Turbo Modula-2 (1 disk) **$89.95**
The best high-level language development system for your Z80-compatible computer. Created by a famous language developer. High performance, with many advanced features; includes editor, compiler, linker, 552 page manual, and more.

Z-COM (5 disks) **$119.00**
Easy auto-installation complete Z-System for virtually any Z80 computer presently running CP/M 2.2. In minutes you can be running ZCPR3 and ZRDOS on your machine, enjoying the vast benefits. Includes 80+ utility programs and **ZCPR3: The Manual**.

Z-Tools (4 disks) **$169.00**
A bundle of software tools individually priced at $260 total. Includes the ZAS Macro Assembler, ZDM debuggers, REVAS4 disassembler, and ITOZ/ZTOI source code converters. HD64180 support.

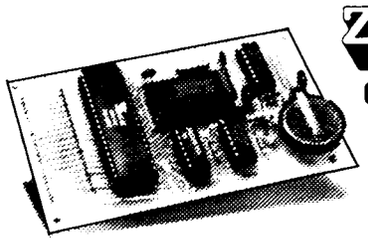
PUBLIC ZRDOS (1 disk) **$59.50**
If you have acquired ZCPR3 for your Z80-compatible system and want to upgrade to full Z-System, all you need is ZRDOS. ZRDOS features elimination of control-C after disk change, public directories, faster execution than CP/M, archive status for easy backup, and more!

DSD (1 disk) **$129.95**
The premier debugger for your 8080, Z80, or HD64180 systems. Full screen, with windows for RAM, code listing, registers, and stack. We feature ZCPR3 versions of this professional debugger.

Quick Task (3 disks) **$249.00**
Z80/HD64180 multitasking realtime executive for embedded computer applications. Full source code, no run time fees, site license for development. Comparable to systems from $2000 to $40,000! Request our free Q-T Demonstration Program.

Z-System OEM inquiries invited.
Visa/Mastercard accepted. Add $4.00 shipping/handling in North America, actual cost elsewhere. Specify disk format.

Echelon, Inc.
P.O. Box 705001-800
South Lake Tahoe, CA 95705 **(916) 577-1105**



Ztime-I
CALENDAR/CLOCK
KIT
Still Only
$69.00

- Works with any Z-80 based computer.
- Currently being used in Ampro, Kaypro 2, 4 & 10, Morrow, Northstar, Osborne, Xerox, Zorba and many other computers.
- Piggybacks in Z80 socket.
- Uses National MM58167 clock chip.
- Battery backup keeps time with CPU power off!
- Optional software is available for file date stamping, screen time displays, etc.
- Specify computer type when ordering.
- Packages available:

Fully assembled and tested	$99.
Complete kit	$69.
Bare board and software	$29.
UPS ground shipping	$ 3.

MASTERCARD, VISA, PERSONAL CHECKS, MONEY ORDERS & C.O.D.'s ACCEPTED.
N.Y. STATE RESIDENTS ADD 7% SALES TAX

KCT KENMORE COMPUTER TECHNOLOGIES
30 Suncrest Drive, Rochester, N.Y. 14609 (716) 654-7356

DosDisk™

An MS-DOS Disk Emulator for CP/M

DosDisk is system software for CP/M 2.2 and CP/M 3 (CP/M Plus) Z80 computers. Unlike any other program, **DosDisk** allows CP/M programs to use files stored on an MS-DOS (PC-DOS) floppy disk *directly* -- without intervening translation or copying.

With **DosDisk**, you can *log into* the pc disk, *including subdirectories*. Regular CP/M programs can read, write, rename, create, delete, and change the attributes of MS-DOS files, just as if they were stored on a CP/M disk. The disk, with any modified files, can immediately be used on a pc.

On DateStamper, QP/M and CP/M 3 systems **DosDisk** automatically stamps MS-DOS files with the current date and time when they are created or modified.

DosDisk supports the most popular MS-DOS format: double-sided double-density 9-sector 40 track disks. It cannot format disks or run MS-DOS programs.

Preconfigured Versions

DosDisk is available for:

all Kaypros with a TurboRom
all Kaypros with a KayPLUS rom and QP/M
Xerox 820-I with a Plus 2 rom and QP/M
Ampro Little Board
SB180 and SB180FX with XBIOS
Morrow MD3
Morrow MD11
Oneac ON!
Commodore C128 with CP/M 3 and 1571 drive

The resident system extension (RSX) version uses about 4.75K of main memory (plus 2K for the command processor). For the SB180 and SB180FX, a banked system extension (BSX) version is also available; it needs about 5K of the XBIOS system memory and *uses no main memory*.

Kit Version

To use the kit version of **DosDisk**, you need *advanced assembly-language experience* in Z80 programming and *technical knowledge of your computer's BIOS*.

If your computer's BIOS adheres precisely to the CP/M 2.2 or CP/M 3 standards and already has the capability of using a general-purpose externally-set format, **DosDisk** can be customized to work with it. You will need to write a special **DosDisk** overlay.

The BIOS must be able to be configured to use the physical parameters of an MS-DOS disk and to use the logical disk parameter header (dph) and disk parameter block (dpb) values supplied by **DosDisk**. The driver code itself (the code that programs the disk controller, reads and writes sectors, etc.) must reside *in the BIOS*.

==========

DosDisk is available directly from the author of DateStamper and BackGrounder ii:

Plu*Perfect Systems
410 23rd St.
Santa Monica, CA 90402

Name: _____

Address: _____

_____

Computer: _____

Operating system: _____

Check Product:

[ ] **DosDisk** preconfigured version ..... $ 30
[ ] **DosDisk** kit version ..... $ 45
[ ] **DosDisk** manual only ..... $ 5
[ ] **DosDisk** BSX and RSX,
for SB180/SB180FX with XBIOS ..... $ 35
(in California, 6.5% sales tax) ..... ____
shipping/handling ..... $ 3

total enclosed ..... $____

DosDisk © Copyright 1987 by Bridger Mitchell

Systematic Elimination of MS-DOS Files

Part 1—Deleting Root Directories & an In-Depth Look at the FCB

by Edwin Thall

Dr. Edwin Thall, Professor of Chemistry at The Wayne General and Technical College of The University of Akron, teaches chemistry and computer programming.

MS-DOS manages programs without requiring you to know the details of the input/output routines. DOS provides a powerful and valuable service, but some of its operations can be fine-tuned. DATACIDE, a utility to help manage MS-DOS files, is presented in this article. The program displays root directory entries and, if requested, calls the DOS function to delete the file. The File Control Block (FCB) plays an important role in the DATACIDE utility and a substantial part of this paper explains how the FCB works in conjunction with DOS. The FCB and DATACIDE do not support subdirectories and Part II will address this topic.

You can readily eliminate an unwanted directory file by means of the DELETE command. To erase TEST.EXT, type:

```
A>DELETE TEST.EXT
```

But what if you wish to rid the directory of 20 files? DOS requires you to type DELETE followed by each file name/ extension 20 times. For file names with common characters, DOS allows the wild card (*). For example, you can eliminate all .BAK extension files with the command:

```
A>DELETE *.BAK
```

This command has two disadvantages: first, you do not have the option of retaining any file with the .BAK extension, and second, you must type DELETE followed by the file name/ extension to erase files without the .BAK extension.

DATACIDE displays the directory file names one at a time and offers you the option to delete, not to delete, or quit the program. DATACIDE can be used on 160K/180K/320K/360K/1200K floppys or the 20M fixed-disk format.

The Program Segment Prefix

DOS recognizes DELETE as the command to remove a file from the directory. But how is the file name passed along to DOS and eventually to the program? The file name needs to be stored in a place in memory agreed upon by both the program and the operating system. As you'll see, this location is the FCB.

Two types of files can be executed directly from DOS: .COM files and .EXE files. .COM files are faster loading and require less memory space, whereas .EXE files can make full use of all the memory segments. In an .EXE file, you can put the program in one 64K segment, the data in a second, the stack in a third, and additional data in a fourth. An .EXE file can utilize up to 256K of memory for a single program, whereas a .COM file is limited to 64K size.

Before DOS passes control to a .COM or .EXE program, it sets up a 256-byte block of code and data. This area, the Program Segment Prefix (PSP), holds vital information such as how to get back to DOS, the addresses of the code that will take control when you press Ctrl-Break, the amount of memory allocated for the program, the file name characters, and the disk drive to be accessed. The PSP occupies the first 256 bytes of a .COM file or the first 256 bytes of the DS segment for an .EXE file. Let's use the Debug utility to peek at the PSP:

```
A>DEBUG
-D 0,FF
```

You should be looking at 256 bytes of an unopened PSP (Figure 1). Table 1 provides an explanation of the PSP locations.

You can find everything you need to know about a DOS file in the part of the PSP referred to as the FCB. This is one place in memory for passing information about disk files between the operating system and the program. The FCB starts at offset 5CH and extends through 80H. All FCB locations and their functions are listed in Table 2. You can enter file names directly by means of the Debug N command. Type NCAPS.COM and then display the FCB:

```
-NCAPS.COM
-D 5C,7F
DS:005C   00 43 41 50                                          .CAP
DS:0060   53 20 20 20 20 43 4F 4D-00 00 00 00 00 20 20 20   S    COM.....
DS:0070   20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00            ........
```

Offset	Function
00-01H	EXE program exit address
02-03H	Count of paragraphs of all system RAM
04H	Reserved
05-09H	DOS function dispatches
0A-0DH	Terminate address
0E-11H	Ctrl-Break exit address
12-15H	Critical error exit address
16-5BH	Used by DOS
5C-80H	FCB
80-FFH	Disk Transfer Area (DTA)

Table 1. The Program Segment Prefix

Offset	Function
5CH	Disk drive (0=default,1=drive A,2=drive B,etc.)
5D-64H	File name (from 1-8 characters)
65-67H	File name extension (from 1-3 characters)
68-69H	Current block number
6A-6BH	Record size (default value 80H)
6C-6FH	File size
70-71H	Date
72-7BH	Reserved by DOS
7CH	Current record number
7D-80H	Random record number

Table 2. The File Control Block

```
DS:0000   CD 20 00 40 00 9A EE FE-1D F0 42 02 CE 18 70 02   M .@..n .pB.N.p.
DS:0010   CE 18 E2 04 42 05 42 05-01 01 01 00 02 FF FF FF   N.b.B.B.........
DS:0020   FF FF FF FF FF FF FF FF-FF FF FF FF CB 18 CA 2A   ............K.J*
DS:0030   CF 18 00 00 00 00 00 00-00 00 00 00 00 00 00 00   N...............
DS:0040   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:0050   CD 21 CB 00 00 00 00 00-00 00 00 00 20 20 20 20   M!K.........
DS:0060   20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20        .....
DS:0070   20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00   ........
DS:0080   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:0090   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:00A0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:00B0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:00C0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:00D0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:00E0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
DS:00F0   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Figure 1. The PSP of an unopened file

```
          +---------+---------+---------+---------+---------+
FILE:     : block 0 : block 1 : block 2 :         : block n :
          +---------+---------+---------+---------+---------+

          +----------+----------+----------+----------+-----------+
BLOCK:    : record 0 : record 1 : record 2 :          : record 127 :
          +----------+----------+----------+----------+-----------+

          +--------+--------+--------+--------+-----------+
RECORD:   : byte 0 : byte 1 : byte 2 :        : last byte :
          +--------+--------+--------+--------+-----------+
```

Figure 2. Organization of files by block, record, and byte

```
-A 80
DS:0080  PUSH  DS           ;SAVE DS
DS:0081  MOV   AX,0         ;SET AX=0
DS:0084  MOV   DS,AX        ;SET DS=0
DS:0086  MOV   BX,0417      ;OFFSET OF CONTROL IN BX
DS:0089  MOV   AL,[BX]      ;CONTENTS OF CONTROL IN AL
DS:008B  OR    AL,40        ;SET CAPS LOCK ON
DS:008D  MOV   [BX],AL      ;STORE MODIFIED CONTROL
DS:008F  POP   DS           ;RESTORE DS
DS:0090  INT   20           ;RETURN TO DOS
```

Figure 3. Assembler code for CAPS.COM

The file name and extension (CAPS.COM) have been stored in the FCB at offsets 5D-67H. The file name always appears in capitals and is adjusted to the left with trailing blanks (20H). The file name extension (offsets 65-67H) follows for further identification. For an unopened file, the disk default value (00) is placed in offset 5CH.

A file is a collection of blocks, with each block containing 128 records. A record may hold from 1 to 32,767 bytes. The relationship of blocks, records, and bytes is illustrated in Figure 2. For files that are accessed from the first record to the last (sequential files), it makes sense to set the record size at one byte. This way, the number of records corresponds exactly to the number of bytes read or written. Operations to read or write a file use the current block number (offsets 68-69H) and current record number (offset 7CH) to locate a particular record. The first block is designated 00, the second is 01, and so on. The Open File operation sets this entry to 00. The default value for the record size (offsets 6A-6BH) is 128 bytes, and if not altered, records of 128 bytes must be read from or written to a file.

The Disk Transfer Area (DTA) is the place in memory where a record is stored on its way to or from a disk file. The DTA has a default size of 128 bytes and is located at offsets 80-FFH. You may have noticed that the FCB goes up to 80H, whereas the DTA starts at 80H. The reason for the overlap is rooted in the development of CP/M. When CP/M changed from using only sequential files to both sequential and random access, there was not enough room for the random record number. The 4-byte field was assigned offsets 7D-80H and, consequently, overlaps one byte of the DTA. For sequential access, this poses no problem because random record numbers are not used. For random access files, however, the DTA has to be moved to a different location to avoid conflict.

Create a Disk File

All disk input/output operations by file name require the DX register to point to the first location in the FCB (offset 5CH) and the AH register to hold the special DOS function (see Table 3). A file (CAPS.COM) will be created, opened, written to, closed, renamed, and finally deleted. You can invoke the DOS function to Create File with the following assembly language routine (omit comments):

```
-A 100
DS:0100  MOV  DX,5C     ;FCB ADDRESS
DS:0103  MOV  AH,16     ;CREATE FILE
DS:0105  INT  21        ;CALL DOS
DS:0107  INT  20        ;RETURN TO DOS
<RETURN> 2X
```

Run the program and then display the FCB:

```
-G
-D 5C,7F
DS:005C   01 43 41 50                                        .CAP
DS:0060   53 20 20 20 20 43 4F 4D-00 00 80 00 00 00 00 00   S    COM........
DS:0070   17 0D 7B 0A 40 00 00 00-00 00 00 00 00 00 00 00   ..{.@...........
```

Note the changes made to the FCB. Drive A is designated 01 (offset 5CH) and the record size assumes the default value (80H). DOS fills in the date (offsets 70-71H), the time (offsets 72-73H), and the File Allocation Table (FAT) entry point (offsets 74-75H). Actually, an entry point into the FAT has not yet been assigned and the value shown is meaningless. Quit Debug and check the directory listing:

```
-Q
A>DIR
```

CAPS.COM should appear in the directory with a file size of zero.

Write to File

The short program in Figure 3 will be written to the CAPS.COM file. When executed, this program sets the Caps-Lock key by turning on bit 6 in RAM address 00417H. Get back to Debug and open the CAPS.COM file.

```
A>DEBUG
-NCAPS.COM
-A 100
DS:0100  MOV  DX,5C
DS:0103  MOV  AH,0F     ;OPEN FILE
DS:0105  INT  21
DS:0107  INT  20
<RETURN> 2X
-G
```

The DOS function to open the file searches the directory for the file name appearing in the FCB. If a match is found, DOS moves the file's information from the directory to the FCB, and the file is ready to be updated. To write one record of 128 bytes to CAPS.COM, enter the program (Figure 3) directly into the DTA (offset 80H) with the Debug A command.

AH register	Function
0FH	Open File
10H	Close File
11H	Search First Match
12H	Search Next Match
13H	Delete File
14H	Read Sequential File
15H	Write Sequential File
16H	Create File
17H	Rename File
21H	Read Random File
22H	Write Random File
23H	File Size
24H	Set Random Field
27H	Read Block
28H	Write Block

Table 3. DOS functions requiring file name in FCB

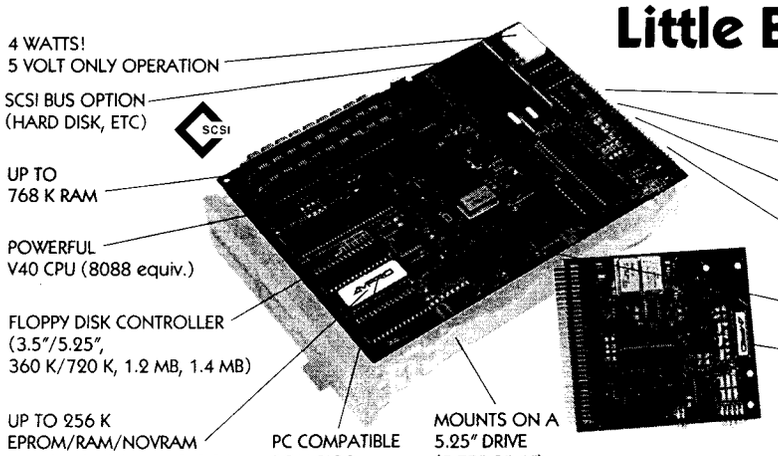
Compact, Low Power, Cost Effective
Single Board Computers
for Embedded Applications

World's smallest PC — and CMOS too!

A Motherboard and 4 Expansion Cards in the Space of a Half-Height 5-1/4" Disk Drive!

from **$359** Qty 1

Little Board™/PC



The CMOS Little Board/PC represents a significant breakthrough in microcomputer technology, providing system designers with a highly compact, self-contained, low power, "PC-compatible" system module in the space of a half height 5-1/4-inch disk drive. Everything but the keyboard, monitor, disk drive, and power supply is included!

The CMOS Little Board/PC is ideally suited for embedded microcomputer applications where IBM PC software and bus compatibility are required and where low power consumption, small size, and high reliability are critical. Its low power requirements, compactness, and solid state disk drive support make the Little Board/PC especially useful in rugged or harsh operating environments.

Typical applications for the Little Board/PC include:

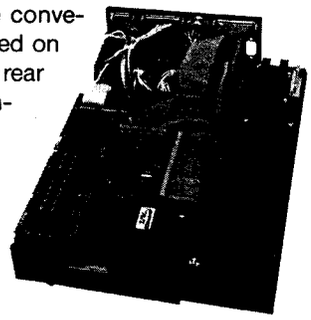
- Data acquisition and control
- Portable instruments
- Protocol conversion
- Telecommunications
- Security systems
- Intelligent terminals
- Diskless workstations
- Remote data logging
- Point-of-sale terminals
- Network servers
- Distributed processing
- SCSI device control

Development Chassis/PC™

The Little Board/PC Development Chassis offers a low cost, "known good" development environment for projects and products based on the Little Board/PC single board computer.

The Little Board/PC Development Chassis includes a two slot PC Expansion Bus, a 360K byte (DSDD) floppy disk drive, a power supply, and all I/O and power cables required for immediate operation with the Little Board/PC.

All I/O connections are conveniently available. Included on the Development Chassis rear panel are standard connectors for keyboard, parallel printer, both serial ports, optional video monitor interface, and the AC power outlet and power switch.



Distributors • Australia: Current Solutions (613) 720-3298 • **Canada:** Tri-M (604) 438-0028 • **Denmark:** Danbit (03) 66 20 20 • **Italy:** Microcom (6) 811-9406 • **Finland:** Symmetric OY 358-0-585-322 • **France:** Egal Plus (1) 4502-1800 • **Germany, West:** IST-Elektronik Vertribes GmbH 089-611-6151 • **Israel:** Alpha Terminals, Ltd. (03) 49-16-95 • **Sweden:** AB Akta 855 0065 • **Switzerland:** Thau Computer AG 41 1 740-41-05 • **UK:** Ambar Systems, Ltd. 0296 435511 • **USA:** Contact Ampro Computers Inc.

AMPRO
COMPUTERS, INCORPORATED

1130 Mountain View/Alviso Road
Sunnyvale, California 94089
(408) 734-2800
TLX 4940302 FAX (408) 734-2939

```
-A 80
```

Before running, remember to change the specification in the AH register from Open File (0FH) to Write File (15H):

```
-E 104
DS:0104  0F.15
-G
-D 5C,7F
DS:005C   01 43 41 50                                          .CAP
DS:0060   53 20 20 20 20 43 4F 4D-00 00 80 00 80 00 00 00   S    COM........
DS:0070   17 0D 26 0F 00 3F 00 00-00 3F 00 00 01 00 00 00   ..&..?...?......
```

One record (128 bytes) was written to the CAPS.COM file and the FCB was updated to reflect the change. Note the file size is now 80H; the time the file was written to has been recorded; the entry point into the FAT filled in at two different locations (offsets 73-74H and 78-79H); and the current record points to the second record (01) in the block. The file must be closed to finalize these changes to the directory. Change the specification in the AH register from Write File (15H) to Close File (10H) and run:

```
-E 104
DS:0104  15.10
-G
-Q
A>DIR
```

CAPS.COM is complete and appears in the directory with a file size of 128 bytes. To test the program, make sure the Caps-Lock key is not set and type:

```
A>caps
```

Rename the File

The renaming of a file necessitates that DOS and the FCB communicate two file names. DOS requires the new file name be stored in offsets 6D-77H. To rename CAPS.COM as CAPSLOCK.COM, use the Debug A command to enter the new file name in the appropriate FCB locations.

```
-NCAPS.COM
-A 6D
DS:006C  DB "CAPSLOCKCOM"
<RETURN> 2X
-D 5C,7F
DS:005C   01 43 41 50                                          .CAP
DS:0060   53 20 20 20 20 43 4F 4D-00 00 80 00 00 43 41 50   S    COM.....CAP
DS:0070   53 4C 4F 43 4B 43 4F 4D-00 3F 00 00 01 00 00 00   SLOCKCOM.?......
```

Call the DOS function to Rename File and then check the directory:

```
-A 100
DS:0100  MOV  DX,5C
DS:0103  MOV  AH,17      ;RENAME FILE
DS:0105  INT  21
DS:0107  INT  20
<RETURN> 2X
-G
-Q
A>DIR
```

Delete the File

When a file is erased, the first character of the file name appearing in the directory is changed to E5H and the file's sectors are rendered free in the FAT. It is not necessary to write the program to perform these operations; one already exists and you can call it with the DOS function. The opening of an existing file is not a prerequisite to deleting a file, but the file name and drive must be placed in the FCB. To erase CAPSLOCK.COM, specify the file name/ drive and call the DOS function to Delete File:

```
A>DEBUG
-NA:CAPSLOCK.COM
-A 100
DS:0100  MOV  DX,5C
DS:0103  MOV  AH,13      ;DELETE FILE
DS:0105  INT  21
DS:0107  INT  20
<RETURN> 2X
-G
```

Value	Description
00H	Normal read®write file
01	Read-only file
02	Hidden
04	System file
08	Volume label
10	Subdirectory
20	Archive bit

Table 4. The attribute bits for file directory entries

A directory search will verify the removal of CAPSLOCK.COM.

```
-Q
A>DIR
```

Extended File Contorl Block

A variation of the FCB, called an extended FCB, can be used to create or access special files. An extended FCB has a 7-byte header (offsets 55-5BH) preceding the 37-byte normal FCB. The first byte of the header must contain 0FFH; the next five locations (offsets 56-5AH) are reserved and not utilized in current MS-DOS versions. The seventh byte (offset 5BH) holds the attribute of the file.

The file attribute byte is needed to identify the type of file (see Table 4). For example, normal files are 00H (if backed-up) or 20H (not backed-up), read only files are designated 01H, and subdirectories are represented by 10H. To demonstrate how to use an extended FCB, the subdirectory EXTENDED.SUB will be created. Modify the empty 7-byte header with the Debug E command and then store the file's name by means of the N command:

```
-E55
DS:0055  00.FF
-E5B
DS:005B  00.10
-NA:EXTENDED.SUB
```

Execute the following short program to create the subdirectory:

```
-A100
DS:0100  MOV  DX,55      ;POINT TO EXTENDED FCB
DS:0103  MOV  AH,0F      ;CREATE FILE
DS:0105  INT  21
DS:0107  INT  20
<RETURN> 2X
-G
-Q
A>DIR
```

Position	Information stored
0-7	File name
8-10	File name extension
11	Attribute
12-21	Reserved
22-25	Time and date
26-27	Entry in FAT
28-31	File size

Table 5. The 32-byte directory field

Format	Sectors	Entries	LSN
160K	4	64	3-6
180K	4	64	5-8
320K	7	112	5-11
360K	7	112	5-11
1200K	14	224	15-28
20M	32	512	17-48

Table 6. The organization of directory sectors

EXTENDED.SUB <DIR> should appear in the directory listing. This file is readily removed with the DOS command:

A>RD \ EXTENDED.SUB

The Datacide Utility

Directory sectors are 512 bytes long and hold 16 file entries. Each file is assigned the 32-byte field listed in Table 5. As you can see, similar information is stored in the directory and FCB. The opening of a file moves data from the directory to the FCB. Modifications are made only to the FCB when a file is updated, however, DOS records changes to the directory when the file is closed.

DATACIDE reads every directory entry and transfers active file names to offsets 5D-67H in the FCB. In order for DATACIDE to know the number of directory sectors to read and where they are located, the disk format must be determined. A special code to identify the disk format is written to the first byte of the FAT during the format operation. These codes are:

F8	fixed disk
F9	1200K
FC	180K
FD	360K
FE	160K
FF	320K

Logical Sector Numbering (LSN) organizes the disk into consecutive sectors beginning with LSN 0. The boot record is always LSN 0, the FAT begins with LSN 1, and directory sectors follow the FAT. Table 6 provides the location of directory sectors by LSN. DATACIDE reads LSN 1 into memory and compares the first byte with F8, F9, FC, FD, FE, and FF. A branch takes place to the appropriate routine when a match is found. If no match is made, the message "DATACIDE CANNOT READ DIRECTORY" is sent to the screen and the program is terminated.

The DATACIDE utility, in assembly code, is listed in Figure 4. The program is organized to:

1. Set the Caps-Lock key so that drive selection and responses of Y, N, and Q are in uppercase.
2. Wait for you to type the directory drive as A, B, or C.
3. Read LSN 1 into memory and identify the disk format.
4. Read the appropriate directory sectors.
5. Set the entry count and move active file names to the FCB. File names are also displayed on the screen.
6. Offer the option to:
   <Y> delete file and read the next entry
   <N> read the next entry
   <Q> quit the program

DATACIDE may on occasion display a few nonsense names at the end of the directory. If this happens, press <Q> to terminate the program. DATACIDE is a handy utility but, with the advent of tree-like file structures, its worth is diminished. The second part of this article will offer a more versatile utility that permits the systematic elimination of all disk files. Part II will also detail the file handle method and demonstrate how to access "secretive" subdirectory files.

Figure 4. Assembler code for DATACIDE.EXE

```
;;;
```

```
        JZ   K160               ;READ SEC 3-6
        CMP  AL,0FCH            ;180K FORMAT?
        JZ   K180               ;READ SEC 5-8
        CMP  AL,0FFH            ;320K FORMAT?
        JZ   K320               ;READ SEC 5-11
        CMP  AL,0FDH            ;360K FORMAT?
        JZ   K320               ;READ SEC 5-11
        CMP  AL,0F9H            ;1200K FORMAT?
        JZ   K1200              ;READ SEC 15-28
        CMP  AL,0F8H            ;HARD DISK?
        JZ   HARD               ;READ SEC 17-48
        MOV  BAIL,0BBH          ;CANNOT READ DIR SECTORS
        MOV  DX,OFFSET MESS5    ;ERROR MESSAGE
        MOV  AH,9
        INT  21H
        RET

K320:   MOV  ENTRIES,112        ;SAVE ENTRY COUNT
        MOV  CX,7               ;READ 7 SECTORS
        MOV  DX,5               ;START WITH SEC 5
        JMP  RSEC               ;READ DIR SECTORS
K160:   MOV  ENTRIES,64         ;SAVE ENTRY COUNT
        MOV  CX,4               ;READ 4 SECTORS
        MOV  DX,3               ;START WITH SEC 3
        JMP  RSEC               ;READ DIR SECTORS
K180:   MOV  ENTRIES,64         ;SAVE ENTRY COUNT
        MOV  CX,4               ;READ 4 SECTORS
        MOV  DX,5               ;START WITH SEC 5
        JMP  RSEC               ;READ DIR SECTORS
K1200:  MOV  ENTRIES,224        ;SAVE ENTRY COUNT
        MOV  CX,14              ;READ 14 SECTORS
        MOV  DX,15              ;START WITH SEC 15
        JMP  RSEC               ;READ DIR SECTORS
HARD:   MOV  ENTRIES,512        ;SAVE ENTRY COUNT
        MOV  CX,32              ;READ 32 SECTORS
        MOV  DX,17              ;START WITH SEC 17
RSEC:   MOV  AL,DRIVE           ;GET DRIVE
        MOV  BX,OFFSET DIR      ;BUFFER AREA
        INT  25H                ;READ DISK
        POPF                    ;RESTORE FLAGS
        RET
RDIR    ENDP
;-------------------------------------------------------
;MOVE FILE NAME TO FCB & DISPLAY ON SCREEN
MOVDIS  PROC NEAR
        MOV  AL,DRIVE           ;GET DRIVE
        INC  AL                 ;INCREASE BY 1 FOR FCB
        PUSH DS                 ;SAVE DS
        PUSH ES                 ;ORIGINAL DS ASSIGNED
        POP  DS                 ;POINTS DS TO FCB
        MOV  BX,5CH             ;DRIVE SPECIFICATION
        MOV  [BX],AL            ;SET DRIVE IN FCB
        POP  DS                 ;RESTORE DS
;MOVE FILE NAME FROM DIR TO FCB
        CALL CRLF
        CALL CRLF
        MOV  SI,OFFSET DIR      ;POINT TO FIRST DIR ENTRY
MOVE:   CMP  ENTRIES,0          ;LAST ENTRY?
        JNZ  CONTIN             ;IF ZERO QUIT PROGRAM
        RET
CONTIN: DEC  ENTRIES            ;NEW ENTRY COUNT
        MOV  AL,[SI]            ;FIRST FILE NAME CHARAC.
        CMP  AL,"A"             ;BELOW "A" ?
        JB   NEXT               ;YES--GET NEXT FILE NAME
        CMP  AL,"Z"             ;ABOVE "Z" ?
        JA   NEXT               ;YES--GET NEXT FILE NAME
        MOV  DI,5DH             ;POINT TO FILE NAME IN FCB
        CLD                     ;SET DIRECTIONAL FLAG
        MOV  CX,11              ;MOVE 11 CHARACTERS
        REP  MOVSB              ;NAME FROM TO FCB
        JMP  SCREEN
NEXT:   ADD  SI,32              ;POINT TO NEXT DIR ENTRY
        JMP  MOVE

;DISPLAY FILE NAME WITH EXTENSION
SCREEN: MOV  CX,8               ;UP TO 8 CHARAC.
        PUSH DS                 ;SAVE DS
        PUSH ES                 ;ORIGINAL DS
        POP  DS                 ;POINTS DS TO FCB
        MOV  BX,5DH             ;FILE NAME IN FCB
FNAME:  MOV  DL,[BX]            ;FILE NAME IN FCB
        CMP  DL," "             ;CHECK FOR BLANK
        JZ   SKIP               ;DISPLAY EXT
        MOV  AH,2               ;DISPLAY CHARAC.
        INT  21H
        INC  BX                 ;NEXT CHARAC.
        LOOP FNAME              ;DO UP TO 8 TIMES

SKIP:   MOV  BX,65H             ;POINT TO EXT IN FCB
        MOV  DL,[BX]
        CMP  DL," "             ;IS EXT BLANK?
        JZ   POPDS              ;NO EXT
        MOV  DL,"."             ;DISPLAY PERIOD
        MOV  AH,2
        INT  21H
        MOV  CX,3
EXT:    MOV  DL,[BX]
        MOV  AH,2               ;DISPLAY EXT CHARAC.
        INT  21H
        INC  BX                 ;NEXT EXT CHARAC.
        LOOP EXT
POPDS:  POP  DS
OPTION:                         ;DISPLAY OPTION MESSAGE
        MOV  DX,OFFSET MESS2    ;YES/NO/QUIT OPTION
        MOV  AH,9               ;PRINT STRING
        INT  21H
;YES/NO/QUIT OPTIONS (MUST SELECT Y, N, or Q)
        MOV  AH,1               ;KEYBD INPUT
        INT  21H
        CMP  AL,"Y"             ;INPUT "Y" ?
        JZ   YES                ;DELETE FILE
        CMP  AL,"N"             ;INPUT "N" ?
        JZ   NO                 ;NEXT DIR ENTRY
        CMP  AL,"Q"             ;INPUT "Q" ?
        JZ   QUIT               ;QUIT PROGRAM
        JMP  OPTION             ;SELECT AGAIN

YES:                            ;DELETE FILE NAME IN FCB
        PUSH DS
        PUSH ES
        POP  DS                 ;ORIGINAL DS
        MOV  DX,5CH             ;POINT TO FCB
        MOV  AH,13H             ;DELETE FILE
        INT  21H
        POP  DS
        CMP  AL,0FFH            ;ERROR CODE
        JZ   ERROR
        MOV  DX,OFFSET MESS3    ;CONFIRM FILE DELETED
        MOV  AH,9               ;PRINT STRING
        INT  21H
        JMP  NO
ERROR:  MOV  DX,OFFSET MESS4    ;OPERATION FAILED
        MOV  AH,9
        INT  21H

NO:                             ;DISPLAY NEXT ENTRY
        CALL CRLF
        CALL CRLF
        ADD  SI,21              ;POINT TO NEXT DIR ENTRY
        JMP  MOVE

QUIT:   RET                     ;END PROGRAM
MOVDIS  ENDP
;-------------------------------------------------------
;CARRIAGE RETURN AND LINE FEED
CRLF    PROC NEAR
        MOV  DL,0DH
        MOV  AH,2
        INT  21H
        MOV  DL,0AH
        INT  21H
        RET
CRLF    ENDP
;-------------------------------------------------------
CSEG    ENDS
;*******************************************************
        END  START
```

■ ■ ■

WordStar 4.0 on Generic MS-DOS Systems

Patching for ASCII Terminal Based Systems

by Phil Hess

As with most software nowadays for MS-DOS systems, MicroPro's WordStar 4.0® assumes IBM compatibility right out of the box. This means that it makes certain assumptions about the system's video adapter memory, hardware ports, and BIOS. MS-DOS® alone is not enough to run WordStar 4.0 as distributed.

However, since WordStar 4.0 is a well-designed program, you can install it for different levels of IBM compatibility by making changes to the user patch area with the WSCHANGE installation program or the MS-DOS DEBUG utility. All patch points are documented in a file named PATCH.LST included on the WordStar distribution disks. This file contains about 40 pages of technical documentation and serves as an excellent guide to the inner workings of WordStar. Anyone who is even remotely interested in learning more about how WordStar works should immediately print out this entire file and spend some time studying it.

One of the first patch points you'll notice in the user patch area listing is the IBMFLG byte. The bits of this flag determine what assumptions WordStar makes about your system, as follows:

Bit 0 -- If set, assumes IBM-compatible BIOS.
Bit 1 -- If set, assumes IBM-compatible timer/counters.
Bit 2 -- If set, assumes IBM-compatible video RAM.
Bit 3 -- If set, assumes IBM-compatible timer tick.

As distributed, all four bits are set. However, if your system isn't completely compatible in one of these areas, you can set the corresponding bit to 0. For example, if writing directly to video RAM causes "snow" on your monitor, you can turn bit 2 off and force WordStar to use BIOS routines for console output. Similarly, if your system lacks the counter/timer which controls the speaker (for sounding a beep) or the timer which generates a user-programmable interrupt every 1/18th second (for printing and editing simultaneously), you can turn those bits off too.

If your system is not even IBM BIOS compatible and runs only generic MS-DOS software, you can turn all the IBMFLG bits off. This forces WordStar to use ordinary operating system functions for console I/O. At this level, all systems which run MS-DOS are compatible.

If the operating system functions are used, you will also need to install WordStar for your terminal. Refer to the PATCH.LST listing and your terminal manual for help with installing the necessary escape codes for cursor addressing, clearing the screen, and so on.

Once installed for generic operation and your particular terminal, WordStar will operate properly, with one exception. Unfortunately, this exception is a rather serious one, and is the primary motivation for this article. Although WordStar will now edit and print files properly, as soon as you try to save an edited file with ^K^D, WordStar dumps you back to DOS. This is indeed perplexing, but has nothing to do with your system. Even when run on an IBM PC and installed for the PC-DOS ANSI console device driver, WordStar does the same thing.

For some reason, WordStar 4.0 has been deliberately crippled to prevent full operation when installed with the IBMFLG BIOS bit set to 0. The reason why MicroPro did this is anyone's guess. Perhaps this was their clumsy way of limiting the range of hardware they would have to support. In any case, there is a simple two-byte patch which forces WordStar to operate correctly, even when installed for a generic MS-DOS system.

This patch is made to the main WordStar program file (WS.EXE) using the MS-DOS DEBUG utility. Since this file is an __.EXE file, it's necessary to rename it using a different extension to force DEBUG to treat the file as a normal file of bytes. Refer to your MS-DOS documentation for help with DEBUG.

Once DEBUG has been loaded, search for the code which is preventing WordStar from saving a file (your responses are underlined):

```
REN WS.EXE WS
DEBUG WS
-S 100 FFFF F6 06 94 01 01 74 1E
8AA6:BC77
```

With my version of WordStar, this code is at location BC77 in the file. With other versions, the location of this code may differ. The pertinent code we're looking for, however, is revealed by disassembling the bytes at this location:

```
-U BC77
8AA6:BC77 F606940101      TEST    BYTE PTR [0194],01
8AA6:BC7C 741E            JZ      BC9C
```

As you can see, this code checks bit 0 (the BIOS bit) of the byte stored at the indicated address (this is where IBMFLG is stored by WordStar at startup). If the bit is 0, WordStar jumps to its exit code, thereby dumping you to DOS. Disabling this jump is all that's needed to make WordStar behave.

Unfortunately, simply substituting NOP's (90 hex) for the two bytes of the JZ instruction isn't enough. While this corrects the ^K^D problem, it creates another problem. Apparently WordStar does a checksum of this portion of the code when it reads in the printer overlay code and acts as though the printer overlay file doesn't exist if the checksum isn't correct. However, there's a way of disabling the bit check without affecting the checksum, as follows (still in DEBUG):

```
-E BC7B 01 1F
-E BC7D 1E 00
```

The effect of this patch is again revealed by disassembling:

```
-U BC77
8AA6:BC77 F60694011F      TEST    BYTE PTR [0194],1F
8AA6:BC7C 7400            JZ      BC7E
8AA6:BC7E                 etc.
```

Now, regardless of the result of the bit test, the JZ will either fall through to the next instruction or jump to it. To offset the change of the JZ relative jump address from 1E to 00, 1E was added to the 01 operand in the previous TEST instruction. This keeps the code's checksum intact.

Once saved to disk and renamed to an __.EXE file, WordStar

will now operate correctly:

```
-W
-Q
REN WS WS.EXE
```

Installing WordStar Using DEBUG

WSCHANGE, WordStar's installation program, itself assumes that your system has an IBM-compatible BIOS. If your system can only run generic MS-DOS software, you will need to install WordStar for your terminal on a different system or else use DEBUG instead of WSCHANGE to install WordStar. However, first try running WSCHANGE on your system. Many MS-DOS systems have some IBM compatibility built into the BIOS, usually enough to run programs that use the BIOS for keyboard input and simple types of console output. Where they usually fall down is in emulating IBM BIOS interrupt 10, functions 6 and 7. These functions perform window scrolling on an IBM video adapter and are difficult to emulate with a serial terminal. WSCHANGE does not use these functions and so should run correctly even with only limited BIOS compatibility.

To use DEBUG to install WordStar, rename WS.EXE as shown above and add 0700 to the patch area address to obtain the DEBUG location. For example, to patch the IBMFLG byte, which is at address 023B in the patch area, use 093B as follows:

```
-E 093B 1F 00
```

Custom Console Status Routine (UCNSTA)

Because MS-DOS does not provide appropriate functions for checking keyboard status, most programs use the BIOS to check if a key has been pressed. You will probably want your patched WordStar to do likewise by writing a custom console status routine which uses your system's BIOS. This routine can be installed at patch point EXTRA (0FB8) in the user patch area. If your system's BIOS has an IBM-compatible interrupt 16 (keyboard input), the following console status routine will work:

```
0FB8   B4 01     EXTRA:  MOV AH,1      ;Function 1 (kbd status)
0FBA   CD 16             INT 16        ;Interrupt 16
0FBC   74 03             JZ  NOTRDY    ;Jump if no char ready
0FBE   B0 FF             MOV AL,FF     ;Return FF if char ready
0FC0   C3                RET
0FC1   B0 00     NOTRDY: MOV AL,00     ;Return 00 if not ready
0FC3   C3                RET
```

Be sure to install a jump to this routine at patch point UCNSTA (0654) in the patch area as per instructions in the patch listing:

```
0654   E9 61 09  UCNSTA: JMP EXTRA
```

In addition, if your system or terminal does not have a type-ahead buffer, set bit 5 of MPMFLG (023D in the user area) to 0. WordStar will then perform more frequent checks of the keyboard's status, providing an adequate type-ahead facility.

Custom Video Attribute (VIDATT) Routine

Another patch you can make is to write a custom video attribute routine for your terminal. This routine is installed at patch point VIDATT (054B) and is fully documented in the user area listing. To summarize, WordStar passes the current video attributes (highlighted, inverse, etc.) to the routine in the CL register and it is up to the VIDATT routine to turn on these attributes.

A simple routine is given below which highlights all marked text (bold, italics, error messages, etc.) and displays normal text in dim. This routine should work on any terminal which uses ESC (and ESC) to display bright and dim text.

```
054B   80F900   VIDATT: CMP CL,00    ;Any attribute bits set?
054E   740D            JZ  DIM      ;Jump if not
0550   B21B            MOV DL,1B    ;Load ESC
0552   B406            MOV AH,06    ;DOS func 6 (console output)
0554   CD21            INT 21       ;Call DOS to output char
0556   B228            MOV DL,28    ;Load (
0558   B406            MOV AH,06    ;DOS function 6
055A   CD21            INT 21       ;Call DOS to output char
055C   C3              RET
055D   B21B     DIM:   MOV DL,1B    ;Load ESC
055F   B406            MOV AH,06    ;DOS func 6 (console output)
0561   CD21            INT 21       ;Call DOS to output char
0563   B229            MOV DL,29    ;Load )
0565   B406            MOV AH,06    ;DOS function 6
0567   CD21            INT 21       :Call DOS to output char
0569   C3              RET
```

Naturally, if your terminal can also display text underlined and in reverse video, you can write a more sophisticated VIDATT routine for on-screen display of those attributes as well.

Extended Characters

If your terminal is not capable of displaying the IBM extended set of foreign and line-drawing characters (ASCII 128-255), you can make additional patches to substitute normal ASCII characters wherever WordStar uses these extended characters. For example, WordStar uses line-drawing characters to draw a box around its menus. If your terminal can't display these characters, you can substitute hyphens or blanks at patch point BOXCHR (0632) in the user patch area.

Similarly, WordStar uses the IBM dot character for displaying soft spaces. You can substitute a normal space or a plus sign for this extended character at patch point SOFTSP (04BC) in the user patch area.

One place where WordStar uses extended characters that are not located in the patch area is in its display of the symbol which represents the Enter key on IBM keyboards. WordStar uses several extended characters to display something which looks like this:

```
<-'
```

This string of three extended characters can easily be found in the WSMSGS.OVR file and replaced by normal ASCII characters. To locate these characters, use DEBUG to search for them:

```
DEBUG WSMSGS.OVR
-S 100 FFFF 11 C4 D9
8AA6:4251
-E 4251 11 3C      (Substitute a <)
-E 4252 C4 2D      (Substitute a -)
-E 4253 D9 27      (Substitute a ')
-W
-Q
```

Before making any of these changes, try running WordStar to see if your terminal can display the IBM extended character set. Many modern terminals such as those from Wyse can display the entire IBM set.

Conclusion

Although MicroPro made it difficult to run WordStar 4.0 on generic MS-DOS systems, with the right patches it's possible to force WordStar to operate correctly. At the same time, it should be pointed out that WordStar is one of the few major software packages that comes with enough information about its inner operation to make these changes possible.

Note: This article was composed with the IBM version of WordStar 4.0 (patched as described above) running on a Morrow MD3 equipped with a Co-Power Plus 8088 co-processor board and Wyse WY-99GT serial terminal. ■

K-OS ONE and the SAGE 68000

Part 2 — System Layout and Hardware Configuration

by Bill Kibler

This is the second of many parts dealing with bringing up the KOS-ONE operating system on a 68000 computer. The first installment covered some general operating system information and specific steps in bringing up the SAGE/KOS-ONE boot loader. In this installment we will cover more on the overall system layout and get into the SAGE hardware configuration.

Making a BIOS for a computer system is a major programming task, and as such requires some planning and organization. In part one we quickly covered some of the considerations, mainly the disk formats. Programming the BIOS requires an understanding of the hardware handshaking. A number of design considerations will be needed by you, as not all of the options will be implemented. Two programming steps must be done first: hardware specification tables; and software flow diagrams.

Hardware Specifications

To understand the project we must first outline the system as built or designed. In this case we are using a SAGE computer and must look at the number and type of devices used. In setting up the BOOT LOADER we just copied code from the SAGE's utilities and made all the calls to the PROM. Our direction now is to make an operating system that will do exactly what we wish it to do, and hopefully somewhat faster than using the PROM. Although using the PROM calls did work and will allow the system to work, the system ran far too slow, taking 2 minutes to boot. One of the reasons I like to make my own systems, is to have a fast boot operation, because as a systems programmer, my systems at first, crash often.

One learns by making mistakes, but not having enough information about your system will most likely keep you from learning at all. When we talked about the disk formats, it was necessary because those were hardware design limits. Ignoring those limits would mean never getting started. The same is true for the BIOS design. Should you decide to put all the code where there is no memory, the system will not work. The hardware sets the limits, and defines the options.

In the SAGE we have 512K of 8 bit memory with parity. The disk controller is a NEC 765. There are two 8255s each providing three 8 bit parallel ports. Interrupts are handled by both the 68000 and an interrupt controller 8259. Two 8253s provide clock and baud rates. Two 8251s provide the terminal and remote serial interfaces.

The Hawthorne TINY GIANT uses a 68681 DUART which provides all of the serial and parallel interfaces. The disk controller is a 1770 by Western Digital. The system has 256 or 512K of 8 bit memory. Three interrupts are used directly with the 68000 as well as TRAP1. The overall design and programming problems are a little easier due to a more straight forward design.

The SAGE's design has multiple layers of interrupts and more items that must be checked and initialized properly. I have included some of the tables and charts that must be created in order to sort this information out. The hardware layout which describes the general overall design is shown in Figure 1. The first item that should catch our eye is the interrupt structure. We see several items that could be ignored, and in my case the optional winchester interface is not present. I doubt that I will use the IEEE-488 interface or all the clock driven interrupts. The items of interest to me are the floppy controller, the terminal serial device, and the real time clock. This does not mean I will not use the extra serial port, just that I will use it in a interrupt mode. In the case of the Centronics port, polling and strobing the device will most likely be more than adequate. It will always be possible at a later date to add these features into the interrupt system.

Shown in Figure 2 is what I call a device map. This map lists the devices, their addresses, in the sample, the bits and their meanings. When we check over the individual bits, we see that several are used to turn other functions on and off. The C port of U22 turns several of the floppy disk operations on and off. The NEC 765 is capable of selecting drives but SAGE chose to do this through the C port. Motors are also turned on by this port. We can reset the controller from here as well. I have also indicated how we want the device to be setup and which bits should be set at the beginning.

Software Flowchart

Now that we have an understanding of the hardware layout, the next step is to determine the design of our software interface. The flow chart of the initialization process shown in Figure 3 is done in a block manner with one section expanded to show the type of detail ultimately needed. This detail will look somewhat like Pascal or any structured language. I once worked with a person who did her design in pseudo-Pascal first, then changed all the ideas into assembly code. The design behind Pascal was exactly that — a method to lay out and design a program. Once properly laid out, any form of language can be substituted for the ideas expressed.

It is at this stage that we determine what and how we will handle various tasks. If we compare the sample BIOS supplied by Hawthorne we can see that it is set up for a 1770 disk controller. This controller uses a separately addressed register for each part of the operation. The 765 has a serialized command structure. That means a series of commands, one after the other, is submitted to the data port. At the end of the command string the chip will then perform the operation. I personally think the 1770 is a better chip, but then I have more experience with that line of devices. The NEC 765 is the device used in IBM PC computers, most likely due to the cost rather than the design abilities.

The difference is less in the serial devices, where it is possible to substitute serial port address for ports in the DUART program. However the 765 requires different handling of the entire command operation. We see this most clearly when comparing the disk read operation steps listed. I have compared a section of pseudo code to show the difference. In the 1770 code we can handle each operation separately. The 765 does however require making up of a table, changing values in it, and then writing it to the command register. I know that most of you will not see much difference,

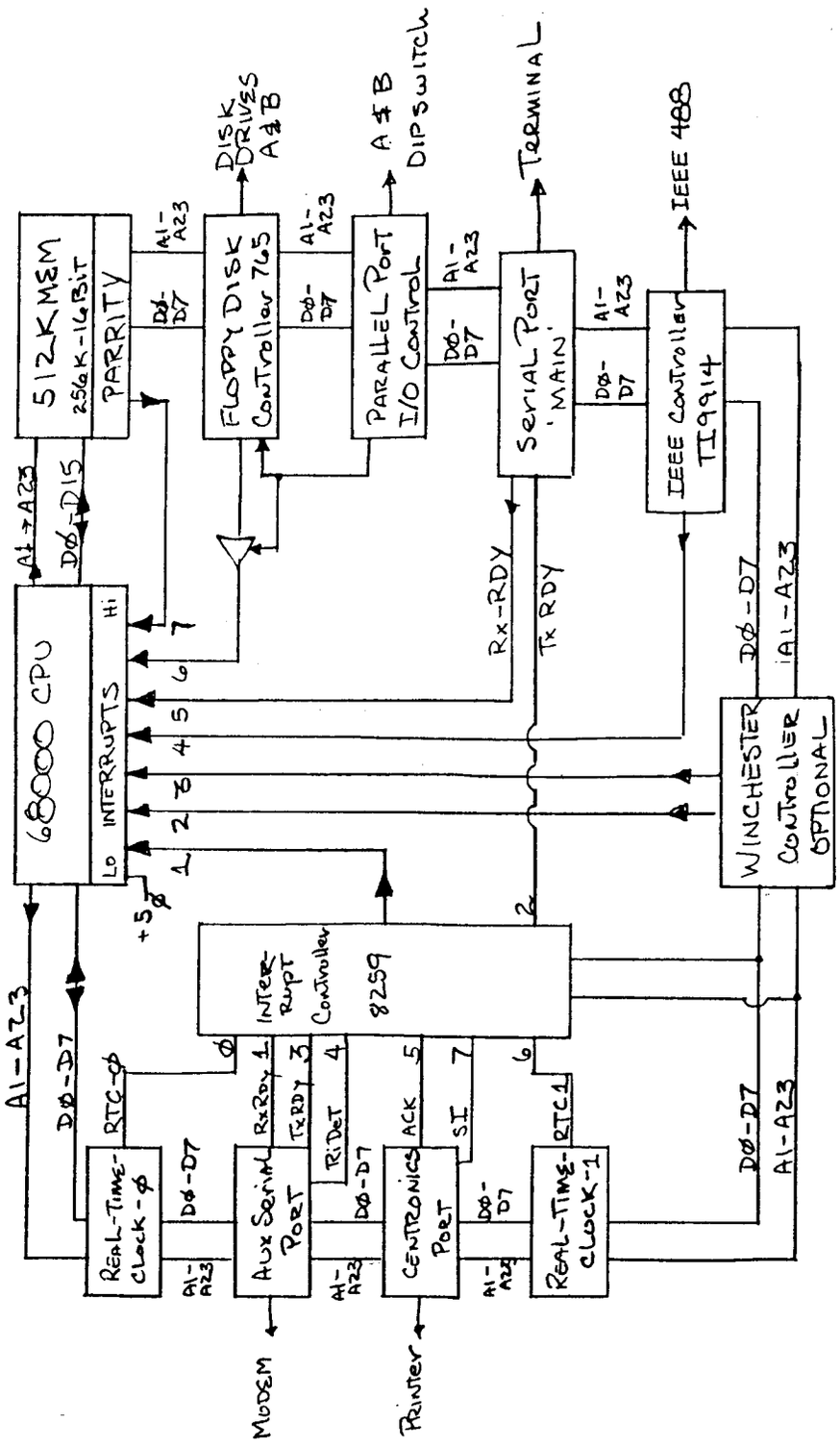


Figure 1:
Hardware Layout

Figure 2:

SAGE II DEVICE MAP

U-22 8255A ADDRESS FFC021-27

DIPSWT FFC021 (DIPSWITCH A)
BIT 0,1,2 = BAUD RATE SELECTION OF TERMINAL PORT
BIT 3 = PARITY OF TERMINAL PORT
BIT 4,5 = BOOTING DEVICE
BIT 6 = 48 OR 96 TPI DRIVES
BIT 7 = BOOT WITHOUT DESTROYING MEMORY

DIPSWT+2 FFC023 (DIPSWITCH B)
BIT 0,1,2,3,4 = IEEE 488 TALK/LISTEN ADDRESS
BIT 5 = TALK ENABLE/DISABLE
BIT 6 = LISTEN ENABLE/DISABLE
BIT 7 = 1 OR 2 ADDRESS

DIPSWT+4 FFC025 (DISKCONTROLLER CONTROL SIGNALS)
BIT 0 = TC+ TERMINATE FLOPPY DMA OPERATION
BIT 1 = RDY+ READY SIGNAL TO FLOPPY
BIT 2 = FDIE+ FLOPPY DISK ENABLE INTERRUPT
BIT 3 = SL0- SELECT DRIVE 0
BIT 4 = SL1- SELECT DRIVE 1
BIT 5 = MOT- TURN MOTOR ON / OFF
BIT 6 = PCRMP- ENABLE PRECOMP
BIT 7 = FRES- FLOPPY DISK RESET SIGNAL

F8255 FFC027 (CONTROL WRITE REGISTER)
10010010B = A=INPUT, B=INPUT, C=OUTPUT
11111000B = SET ALL BITS TO THEIR NON ACTIVE LEVEL

U-39 8259A ADDRESS FFC061-67

C8255 FFC061 (CENTRONICS DATA PORT)
BIT 0-7 = DATA BITS 0 TO 7 OF J8-CENTRONICS

C8255+2 FFC063 (CENTRONICS AND FLOPPY INPUT)
BIT 0 = FDI+ FLOPPY INTERRUPT OUT
BIT 1 = WP+ DRIVE IS WRITE PROTECTED
BIT 2 = RG- RING DETECTED MODEM
BIT 3 = CD- CARRIER DETECTED MODEM
BIT 4 = BUSY- PRINTER BUSY J8 PIN 22
BIT 5 = PAPER- PAPER OUT J8 PIN 23
BIT 6 = SELECT- SELECT STATUS J8 PIN 24
BIT 7 = FAULT- PRINTER FAULT J8 PIN 28

C8255+4 FFC065 (CENTRONICS AND UTILITIES)
BIT 0 = PRES- PARITY ERROR RESET
BIT 1 = SC+ IEEE488 STROBE
BIT 2 = SI+ SOFTWARE INTERRUPT 7
BIT 3 = LEDR+ STATUS LED ON FRONT PANEL
BIT 4 = STROBE- PRINTER STROBE J8 PIN 1
BIT 5 = PRIME- PRINTER PRIME SIGNAL J8 PIN 26
BIT 6 = CLEAR TO ACK- INTERRUPT FLIPFLOP
BIT 7 = RMI- RESET MODEM INTERRUPT FLIPFLOP

SI8255 FFC067 (CENTRONIC CONTROL PORT WRITE ONLY)
10000010B = A=OUT, B=INPUT, C=OUT
00110000B = STROBE AND PRIME SET HI

U-73 8259-5 ADDRESS FFC041

P8259 FFC041 (INTERRUPT CONTROLLER)
IR0 = REAL TIME CLOCK 8253 OUT 2
IR1 = RX2I+ MODEM REC READY
IR2 = TX1I+ TERMINAL TX READY
IR3 = TX2I+ MODEM TX READY
IR4 = MI- MODEM RING/CD INTERRUPT FLIPFLOP
IR5 = CNI+ CENTRONIC ACK- FLIPFLOP
IR6 = REAL TIME CLOCK 8253 OUT 0
IR7 = SI+ SOFTWARE INTERRUPT

P8259 FFC041 (CONTROL PORT FOR INIT)
00010010B

P8259+2 FFC043 (CONTROL PORT WRITE ONLY)
00000000B = NO ADDRESS BITS
00110000B = MASK BITS IR4&5 (NO INTERRUPTS ALLOWED)

U-36 LS148 8 TO 3 ENCODER (68K USES CODED INTERRUPT)

VECTOR	BITS	IR#	
07C	111	7	MEMORY PARITY ERROR HIGHEST PRIORITY
078	110	6	FLOPPY DISK GATED INTERRUPT = TRAP6
074	101	5	TERMINAL RECIEVE READY = TRAP5
070	100	4	IEEE 488 INTERRUPT
06C	011	3	WINCHESTER INTERRUPT
068	010	2	WINCHESTER INTERRUPT
068	001	1	INTERRUPT CONTROLLER = TRAP1
060	000	0	+5V NO INTERRUPT ALLOWED

OTHER DEVICES ADDRESSES

PROM	FE0000-FE3FFF	PROM MEMEORY LOCATION
I/O	FFC000-FFC3FF	I/O DEVICES
BRATE	FFC001	REAL TIME CLOCK
BAUD1	FFC003	SERIAL PORT TERMINAL BAUD RATE
BAUD2	FFC005	SERIAL PORT MODEM BAUD RATE
BAUDC	FFC007	CONTROL PORT OF 8253S COUNTER
IEEE	FFC011-FFC01F	IEE488 PORTS (TMS 9914)
DIPSWT	FFC021-FFC025	DIPSWITCH AND FLOPPY DATA PORT
F8255	FFC027	CONTROL PORT OF DIPSWITCHS
REMIO	FFC031	SERIAL MODEM PORT DATA
REMCS	FFC033	MODEM CONTROL/STATUS PORT 8251A
P8259	FFC041-FFC043	INTERRUPT CONTROLLER 8259
FDMSR	FFC051	FLOPPY DISK STATUS PORT NEC765
FDDATA	FFC053	FLOPPY DATA/COMMAND PORT
C8255	FFC061-FFC065	CENTRONICS AND SYSTEM SIGNAL PORT
SI8255	FFC067	CONTROL PORT OF CENTRONICS 8255
TERMIO	FFC071	SERIAL TERMINAL PORT 8251A
TERMCS	FFC073	SERIAL TERMINAL CONTROL PORT
TIMER	FFC081-FFC087	REAL TIME CLOCK

Figure 3:

BIOS FLOW CHART

START

INITIALIZE
HARDWARE

SET
VARIABLES

LOAD
"OPERATE.BIN"
"COMMAND.BIN"

SET
INTERRUPTS

GOTO
OPERATE.BIN

ON ERRORS
GO MONITOR

INTERRUPT HANDLERS

ADDRESS 068h AUTOVECTOR 1
  INTERRUPT CONTROLLER ROUTINES
    0 = UPDATE REAL TIME CLOCK COUNTER
        WITH 1/10 SEC COUNT. CHECK
        MOTOR ON FLAG, TURN OFF IF TIMED
        OUT, ELSE UPDATE COUNT.
    2 = LOAD TERMINAL TX PORT IF DATA IN
        TERMINAL OUTPUT QUEUE.
    5 = PRINTER ACK RECIEVED, RESET ACK &
        STROBE; CHECK FOR DATA IN QUEUE AND
        OUTPUT (background spooling)
    * = ALL OTHER INTERRUPTS TO BE MASKED
        OFF AND IGNORED.

ADDRESS 074h AUTOVECTOR 5
  TERMINAL RECIEVE READY INTERRUPT
    LOAD CHARACTER FROM TERMINAL IN PORT
     INTO QUEUE AND UPDATE QUEUE POINTER.

ADDRESS 078h AUTOVECTOR 6
  FLOPPY DISK GATED INTERRUPT
    SIGNALS END OF FLOPPY OPERATION,
     CHECK FOR ERRORS AND SET FLAGS.

ADDRESS 07Ch AUTOVECTOR 7
  PARITY MEMORY ERROR
    RESET PARITY CIRCUIT
    JUMP MONITOR

ADDRESS 084h TRAP 1 VECTOR
  CHANGE FROM USER TO SYSTEM STATE
    SAVE ALL USER STACKS AND REGISTERS
     RESTORE SYSTEM STACKS, POINTERS AND
     REGISTERS.

ALL OTHER VECTORS
  INDICATE VECTOR NUMBER AND PRINT TO SCREEN
  JUMP TO MONITOR FOR ERROR HANDLING.

KOS-ONE BIOS TO OPERATE.BIN JUMP TABLE

OSINIT.L
  INITIALIZE SYSTEM POINTERS AND STACKS
USER.L
  SAVE USER STACKS AND REGISTERS, THEN GOTO
   USER PROGRAM.
CLOCKSET.L
  SET REAL TIME CLOCK.
EXSET.L
  SET EXCEPTION VECTORS (NOT IMPLIMENTED)
WAIT.L
  WAIT NUMBER OF MILLISECONDS ON TOP OF STACK.
GETCON.L
  GET CHARACTER FROM TERMINAL IN QUE.
PUTCON.L
  PUT CHARACTER INTO TERMINAL OUT QUE.
INITCON.L
  RESET TERMINAL SERIAL DEVICE.
STATCON.L
  RETURN STATUS OF TERMINAL.
CNTRLCON.L
  CHANGE PARAMETERS OF TERMINAL (NOT USED).
GETPRN.L
  GET CHARACTER FROM PRINTER (X-ON/X-OFF; NOT USED)
PUTPRN.L
  OUTPUT CHARACTER TO PRINTER PORT AND STROBE
   PRINTER DEVICE.
INITPRN.L
  SET UP PARALLEL PORT FOR CENTRONICS PRINTER.
STATPRN.L
  CHECK FOR PRINTER NOT ON LINE.
CNTRLPRN.L
  CHANGE FROM CENTRONICS TO SERIAL PRINTER.
DISKREAD1.L
  READ GIVEN SECTOR FROM DISK.
DISKWRITE1.L
  WRITE GIVEN SECTOR TO DISK.
DISKSTAT1.L
  SEE IF DISK IS BUSY.
DSKCNTRL1.L
  PERFORM SPECIFIED ACTION ON DRIVE
   IE: RESTORE DRIVE OR FORMAT DISK.
DSKFLSH1.L
  FLUSH DISK BUFFERS IF USED (NOT USED).

but to me it is a lot of extra programming steps to achieve the same operation.

From a strictly hardware point the 765 has some other features that have caused many problems. The device as designed polls the drives at all times. This means it is checking to see if the drive status has changed (did the drive door open?), and that the drive hardware must be ready at all times. In the case of 8 inch drives this can produce considerable amounts of heat that must be removed. While working at one computer company, their systems used the 765 as designed and many users had heat problems as well as hardware problem from this chip. SAGE has gotten around this problem by using an external ready signal (not the disk drives) and selecting drives with the parallel port. This again takes extra programming operations to handle the non-standard hardware utilization.

Putting it Together

The next step, which to me is not as exciting, is writing the code. I enjoy working out the problem in great detail and prefer to turn it over to some one else for the tedious code writing. In this project I get to do both for a change. To help out with the task we can still use the sample BIOS provided, but by using our charts and tables we make only those changes needed. Out of a hundred lines of initialization code, I only changed a third of it. Most of the structure remains the same, just the actual device specific items are different. When I left you last time, I had tried to indicate a number of pitfalls to consider. By using the sample BIOS and being careful of which code gets changed we should keep those problems to a minimum.

The problems of which I speak are register utilization. The HTPL operating system is a stack oriented program. As such it depends on several registers for the locations of those stacks. Status is also passed on those stacks. If one messes with the wrong address register, loss of the stack might occur (system crashes). My first time through the book on KOS-ONE left me with too vague a feeling for register usage. It was not until I had experienced some crashes and solved them by protecting the pointer registers that it all made sense. The manual explains some of this in the assembly language section.

One method I used for clarifying the operation, was a table for each type of entry. This indicates which register contains what, and whether it was pushed as a word or long. We must keep in mind that words and longs are handled somewhat differently. If data is pushed as a long, it must be popped as a long. Not doing so will result in dire consequences. You will also need to know how much is expected back on each stack when you return from

```
        REPEAT ABOVE FIVE DISK FUNCTIONS FOR
             DRIVES 2,3, AND 4.
        GETAUX.L
             GET CHARACTER FROM MODEM PORT
        PUTAUX.L
             SEND CHARACTER TO MODEM PORT.
        INITAUX.L
             SET BAUD RATE FOR MODEM PORT.
        STATAUX.L
             CHECK FOR DTR, RING, READY.
        CNTRLAUX.L
             CHANGE OTHER PARAMETERS OF MODEM.
        MONITOR.L
             JUMP INTO MONITOR.

    DETAILS OF DISKREAD OPERATION FOR 765 CONTROLLER
         ( in pseudo code )

process: DISKREAD ( (A4.L)REC#,(A4.L)BUFADR -> (A4.L)ERRFLG )

      GET drive number, set/store DRIVE(A2)
      POP Read Buffer Address, store BUFPN(A2)
      POP logical record number, store RECORD(A2)
      BSR TRANSFORM (change record number to sector, side, track )
           Convert record to sector number ( rec mod sec + 1 )
           Get track number (rec/(totside * totsect))
           Get side number (rec mod((totside * totsect)/totsect))
           If track >= tottrack then error
           insert data into command string
           end transform
      TST.B for error
           if error do READERRORS and EXIT
      BSR SEEK (move head to requested drive, track, and side )
           BSR SURFACE (select drive and track )
                Compare drive # to Current drive # (DRIVE(A2)=CURRDRV(A2))
                If = then seek track (TRACK(A2))
                     else select/unselect drive & side
                end surface
           BSR CYLINDER
                check controller for activity, wait if yes
                fetch track number
                prepare command string for seek
                execute seek command
                check status of seek
                     if error set flags
                end cylinder
           if errors
                restore drives
                     issue restore comand string
                     reseek and test for proper track
                if restore ok jump cylinder
                     else restore again
                     else error out
                store current track and drive information
           else store data and end SEEK
      TST.B for errors
           if errors do READERRORS and EXIT
      BSR READSEC (now do actual reading of sector data )
           check for busy controller
                if busy wait loop
           load A0 with address of data
           load sector number to read into command string
           enable interrupts for floppy action
           do READCOMMAND
                check for status of read
                if not complete continue loop
                if not ok read do RETRYS
           end READSEC
      TST.B for errors
           if errors do READERRORS and EXIT
      SET successful read flags and EXIT

      READERRORS
           SET unsucessful read flags and EXIT
```

```
EXIT
     RESTORE stacks and registers
     execute RTS ( return to BDOS )

END DISKREAD
```

the operation. Not having the right number of items on a stack is certain disaster.

Till Next Time

While dealing with the KOS-ONE program I have noticed a few concerns. Joe has not had enough time to fully document every thing and some items remain a mystery. A few bugs are present in the system, although they are minor and are being fixed. I have had considerable trouble with the drive selection, until Joe reminded me to use the CHDIR command to read the other drive.

In any case next time I will fill in any unanswered questions and hopefully get into the HTPL code. ■

SuperBASIC 68K

by Custom Computer Products

A BASIC FOR THE K-OS ONE OPERATING SYSTEM

* LOAD and RUN most standard MS BASIC programs without change.
* Integer, single precision floating point and 64 bit double precision floating point for arithmatic and called functions.
* Variable names can be any length up to 255 characters, all of them significant.
* A full set of string functions is included. Strings can be any length up to 255 chars.

SuperBASIC 68K is the newest software package available for the K-OS ONE operating system. It is a small (approx 32K), full featured BASIC that can utilize all available memory.

For a full specification, contact Hawthorne Technology.

K-OS ONE OPERATING SYSTEM

Get the K-OS ONE operating system for your 68000 hardware. With it you can read and write MS-DOS format diskettes on your 68000 system. Included in the package are: K-OS ONE w/source code, Editor, Assembler, HTPL Compiler Sample BIOS Code.

68000 SOFTWARE

* K-OS ONE operating system uses MS-DOS disks w/source code $50
* K-OS ONE manual . . . $10
* HT68K SBC w/K-OS ONE $395
* Screen Editor Toolkit $50
* HT-FORTH $100
* BASIC $149

Free Newsletter & Spec Sheet

HAWTHORNE TECHNOLOGY
1411 SE 31st, Portland, OR 97214
(503) 232-7332

Call For Papers

TCJ is establishing a forum on the following areas, and we welcome your submissions and proposals. Candidates for membership in the peer review and advisory groups, including group coordinators, will also be considered.

• **Education in the Next Decade** — Our contacts with both the educators who are preparing the curriculums and the people in industry who need to employ workers with the necessary skills, indicate that the requirements are changing. Industry sources say that current graduates do not have the knowledge to fill available real world positions, and the educators say that they do not have the course material and specific requirements needed to implement the courses. TCJ invites papers from both Academia and Industry to discuss the problem and propose solutions.

• **Language Development** — There is a great need for language development in the areas of command parsers, user interfacing, custom languages, ROM based embedded controller systems, etc. We need papers covering both the theoretical and practical aspects from the viewpoints of both the developers and the users.

• **Database Development** — The commercial programs are very powerful, and there are good texts which explain the commands and functions. What is missing is tutorials on the concepts of the practical aspects of designing and developing a database — the nitty-gritty details on implementing a database rather than an explanation of the tools.

There is also a need for papers on using high level languages to replace or supplement DBMS programs where it is easier or more efficient to perform some of the operations outside of the DBMS.

Other suggested topics are welcome. Query regarding book or monograph manuscripts.

The Computer Journal
190 Sullivan
Columbia Falls, MT 59912
(406) 257-9119

Back Issues Available:

Issue Number 1:
- RS-232 Interface Part One
- Telecomputing with the Apple II
- Beginner's Column: Getting Started
- Build an "Epram"

Issue Number 2:
- File Transfer Programs for CP/M
- RS-232 Interface Part Two
- Build Hardware Print Spooler: Part 1
- Review of Floppy Disk Formats
- Sending Morse Code with an Apple II
- Beginner's Column: Basic Concepts and Formulas

Issue Number 3:
- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for the Apple II
- Modems for Micros
- The CP/M Operating System
- Build Hardware Print Spooler: Part 2

Issue Number 4:
- Optronics, Part 1: Detecting, Generating, and Using Light in Electronics
- Multi-User: An Introduction
- Making the CP/M User Function More Useful
- Build Hardware Print Spooler: Part 3
- Beginner's Column: Power Supply Design

Issue Number 8:
- Build VIC-20 EPROM Programmer
- Multi-User: CP/Net
- Build High Resolution S-100 Graphics Board: Part 3
- System Integration, Part 3: CP/M 3.0
- Linear Optimization with Micros

Issue Number 14:
- Hardware Tricks
- Controlling the Hayes Micromodem II from Assembly Language, Part 1
- S-100 8 to 16 Bit RAM Conversion
- Time-Frequency Domain Analysis
- BASE: Part Two
- Interfacing Tips and Troubles: Interfacing the Sinclair Computers, Part 2

Issue Number 15:
- Interfacing the 6522 to the Apple II
- Interfacing Tips & Troubles: Building a Poor-Man's Logic Analyzer
- Controlling the Hayes Micromodem II From Assembly Language, Part 2
- The State of the Industry
- Lowering Power Consumption in 8" Floppy Disk Drives
- BASE: Part Three

Issue Number 16:
- Debugging 8087 Code
- Using the Apple Game Port
- BASE: Part Four
- Using the S-100 Bus and the 68008 CPU
- Interfacing Tips & Troubles: Build a "Jellybean" Logic-to-RS232 Converter

Issue Number 17:
- Poor Man's Distributed Processing
- BASE: Part Five
- FAX-64: Facsimile Pictures on a Micro
- The Computer Corner

Interfacing Tips & Troubles: Memory Mapped I/O on the ZX81

Issue Number 18:
- Parallel Interface for Apple II Game Port
- The Hacker's MAC: A Letter from Lee Felsenstein
- S-100 Graphics Screen Dump
- The LS-100 Disk Simulator Kit
- BASE: Part Six
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 1
- The Computer Corner

Issue Number 19:
- Using The Extensibility of Forth
- Extended CBIOS
- A $500 Superbrain Computer
- BASE: Part Seven
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 2
- Multitasking and Windows with CP/M: A Review of MTBASIC
- The Computer Corner

Issue Number 20:
- Designing an 8035 SBC
- Using Apple Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering and Other Strange Tales
- Build a S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K
- The Computer Corner

Issue Number 21:
- Extending Turbo Pascal: Customize with Procedures and Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition and Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC
- The Computer Corner

Issue Number 22:
- NEW-DOS: Write Your Own Operating System
- Variability in the BDS C Standard Library
- The SCSI Interface: Introductory Column
- Using Turbo Pascal ISAM Files
- The AMPRO Little Board Column
- The Computer Corner

Issue Number 23:
- C Column: Flow Control & Program Structure
- The Z Column: Getting Started with Directories & User Areas
- The SCSI Interface: Introduction to SCSI
- NEW-DOS: The Console Command Processor
- Editing The CP/M Operating System
- INDEXER: Turbo Pascal Program to Create Index
- The AMPRO Little Board Column
- The Computer Corner

Issue Number 24:
- Selecting and Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assembly Code for CP/M
- The C Column: Software Text Filters
- AMPRO 186 Column: Installing MS-DOS Software
- The Z Column
- NEW-DOS: The CCP Internal Commands
- ZTIME-1: A Realtime Clock for the AMPRO Z-80 Little Board
- The Computer Corner

Issue Number 25:
- Repairing & Modifying Printed Circuits
- Z-Com vs Hacker Version of Z-System
- Exploring Single Linked Lists in C
- Adding Serial Port to Ampro Little Board
- Building a SCSI Adapter
- New-DOS: CCP Internal Commands
- Ampro '186: Networking with SuperDUO
- ZSIG Column
- The Computer Corner

Issue Number 26:
- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside AMPRO Computers
- NEW-DOS: The CCP Commands Continued
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS
- The Computer Corner

Issue Number 27:
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis and Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program for Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying The CP/M Disk Parameter Block for Foreign Disk Formats
- The Computer Corner

Issue Number 28:

- Starting Your Own BBS: What it takes to run a BBS.
- Build an A/D Converter for the Ampro L.B.: A low cost one chip A/D converter.
- The Hitachi HD64180: Part 2, Setting the wait states & RAM refresh, using the PRT, and DMA.
- Using SCSI for Real Time Control: Separating the memory & I/O buses.
- An Open Letter to STD-Bus Manufacturers: Getting an industrial control job done.
- Programming Style: User interfacing and interaction.
- Patching Turbo Pascal: Using disassembled Z80 source code to modify TP.
- Choosing a Language for Machine Control: The advantages of a compiled RPN Forth like language.

Issue Number 29:

- Better Software Filter Design: Writing pipable user friendly programs.
- MDISK: Adding a 1 Meg RAM disk to Ampro L.B., part one.
- Using the Hitachi HD64180: Embedded processor design.
- 68000: Why use a nes OS and the 68000?
- Detecting the 8087 Math Chip: Temperature sensitive software.
- Floppy Disk Track Structure: A look at disk control information & data capacity.
- The ZCPR3 Corner: Announcing ZCPR33 plus Z-COM Customization.
- The Computer Corner.

Issue Number 30:

- Double Density Floppy Controller: An algorithm for an improved CP/M BIOS.
- ZCPR3 IOP for the Ampro L.B.: Implementing ZCPR3 IOP support featuring NuKey, a keyboard re-definition IOP.
- 32000 Hacker's Language: How a working programmer is designing his own language.
- MDISK: Adding a 1 Meg RAM disk to Ampro L.B., part two.
- Non-Preemptive Multitasking: How multitasking works, and why you might choose non-preemptive instead of preemenptive multitasking.
- Software Timers for the 68000: Writing and using software timers for process control.
- Lilliput Z-Node: A remote access system for TCJ subscribers.
- The ZCPR3 Corner
- The CP/M Corner
- The Computer Corner

Issue Number 31:

- Using SCSI for Generalized I/O: SCSI can be used for more than just hard drives.
- Communicating with Floppy Disks: Disk parameters and their variations.
- XBIOS: A replacement BIOS for the SB180.
- K-OS ONE and the SAGE: Demystifing Operating Systems.
- Remote: Designing a remote system program.
- The ZCPR3 Corner: ARUNZ documentation.

TCJ ORDER FORM

Subscriptions		U.S.	Canada	Surface Foreign	Total
6 issues per year					
☐ New ☐ Renewal	1 year	$16.00	$22.00	$24.00	
	2 years	$28.00	$42.00		
Back Issues		$3.50 ea.	$3.50 ea.	$4.75 ea.	
Six or more #'s		$3.00 ea.	$3.00 ea	$4.25 ea.	
				Total Enclosed	

All funds must be in U.S. dollars on a U.S. bank.

☐ Check enclosed ☐ VISA ☐ MasterCard Card # _____

Expiration date_____ Signature _____

Name _____

Address _____

City_____State_____ZIP _____

The Computer Journal

190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406) 257-9119

(Continued from page 44)

ternally was used for such a need, but mine didn't have any of the interface. The schematics I have, show a network interface and I simply modified that design for the Centronics driver.

I have included both the code and the schematic for a simple Superbrain Centronics port. It was also great fun to stop doing 68K assembly and go back to the Z80 code. I certainly like 68K assembly for its straight forward mnemonics, but there still is nothing as enjoyable as Z80 programming. I still feel that it is the most optimized code out. The right number of register, not too many or too little, with just the right number of special indirect and I/O options.

As you look at the schematic you will see that there is very little to it, and in fact that is one of the reason for including it. I feel that some people think there is a lot to making a Centronics port and may not try it. It however is probably the simplest and easiest of projects for a beginner to undertake. As can be seen, this is a simple project that will build up your skills and confidence.

Another 68000 System

I have added a new 68000 based system, the Tandy Model 16. This is an older version that I picked up for $200 including their Xenix® operating system. It is a 8 inch drive system but the Xenix will not work very good without a hard disk. I got it for use with the KOS-ONE operating system. What I have discovered is how they designed it — basically a Z80 model 3 with a master 68K CPU card. The system comes up as a Model 3 and then can load the 68K operating system that uses all the Z80 I/O. That was a very commen approach when this was designed, but they are never as fast as ones designed with the 68K from scratch. The Xenix operating system turned out to be not worth the trouble of learning, which was the other reason for getting this system.

Xenix is Microsoft's version of Unix® for smaller systems. Unix is quite popular with some industry and education people because of the utilities it provides. The system is designed primarily for multiple users and is far too complex for single users. From what I can tell, all the functions are programs that reside on the main boot disk, and you need fast access to a hard disk for operation of the system. It must also be a large disk because the system is constantly writing to the disk. It is a clock based system that stores just about everything that is happening, including every one of your keystrokes. I think that might be fine if you want to make use of that information, but for single users, forget it.

That is one of the reasons that I like and am still working on the single user system of KOS-ONE. Most of the operating systems for the 68K are all based on the Unix idea of multiusers and as such have too much overhead for a single user. Now KOS-ONE is not the only operating system for the 68K, in fact I just received my copy of SKDOS68K. I have not had a chance to bring it up yet, but the manuals are probably the best I have seen in a long time. The cost of SKDOS is higher than KOS-ONE but then the quality of the manuals is worth the extra money.

I must say that I am working with Joe on improving the manuals for the KOS-ONE, and there are other variables in the two systems. The KOS-ONE is intended to be brought up from an PC system. SKDOS is intended to be brought up from a FLEX system. I have been provided PC compatible disk for SKDOS and will be writing on how to bring it up from that system. The main thing to remember is that both of these operating systems can be purchased with hardware. That means you can have a 68K running for less than $500.

If you are interested in learning Unix, there are two other ways you can do that, both are C based Unix look alikes. There are several programs in public domain that will do some Unix like functions, but two products are available for around a $100 each. The first product out was XINU or Unix spelled backwards. This product came out some time ago as a book on Unix with their own system in C code provided (written by Douglas

Comer: *Operating System Design, The XINU Approach*). It has not been too popular as it still requires a large hard disk. The code is suppose to be somewhat close to the real thing and as such has the same design parameters.

The other product was developed for students to use in understanding operating system. It was written by Andrew Tanenbaum and is called MINIX (the book is *Operating Systems:Design and Implementation* by Prentice-Hall). This product is a lot like Unix but has been written to take advantage of the PC system. The code and operating system will work on a PC and may be possible to port over to 68K systems. I understand that this will not be a trivial task as the C code makes considerable use of the library calls. I believe the library source code is not included so changing those routines would be a major problem. If I have not seen other articles on these systems by the end of the year, I will get a copy of MINIX and play around with it myself.

Forth National Convention

In November I attended the Saturday portion of the Forth National Convention in San Jose. The convention covered both Friday and Saturday seminars, but most people could only show up on Saturday. Some of those who were there last year said this was a very small showing. The speakers were very good and had lots to say about how Forth was the only language they found to do what they were doing. A typical example was the college teachers who use it for robotics classes. They take engineering students and teach them Forth while building robotics systems. With time a premium, Forth is the only language they can learn fast enough to be able to also produce the final operating product.

While there I picked up some ROMs and schematics to make my NOVIX 4000 into a full blown computer. The NOVIX chip is so fast it can read floppy disk, take keyboard data, and drive a PC monitor with a total chip count of 17 devices. Most of the work is done in the NOVIX with the other devices being mostly the 16 bit wide memory and clock circuits. I don't have mine running yet, but as soon as I can get to it, I will put it in a Plexiglass box, to show people just how little it takes to make a complete system.

The next year should be exciting for Forth in robotics, as several companies are making Forth engines. These engines are like the NOVIX in that they run Forth directly. All that is needed now is for the computer industry to open up some and look at the alternatives that Forth engines provide. The speed of bringing up systems and changing designs far exceed any other products currently available. What is needed however is more people who know Forth and understand it, which I feel is not happening. I feel personally that so much pressure is being put on the 'C' language that other ways of solving problems are being overlooked.

Ending Comments

Things have been rather busy and I am trying to put more time into these articles but I must keep my sights clearly in front of me. The KOS-ONE and SKDOS68K are my main efforts right now. I feel I am getting a handle on KOS and I plan on checking out SKDOS. I feel good about both of these product as they will help bring more attention to the 68K. What I like the most is the idea of making operating systems that fit your special needs and ways of doing things. I have gotten pretty tired of other programmers thinking they know how I think and work. It is about time that I can create systems that work for me and not against me. ■

Advertiser's Index

AMPRO Computers 28
Austin Code Works 9
C User's Group 6
C.C. Software 19
Computer Journal 40, 41
Echelon, Inc. 2, 24
Hawthorne Technology 39
Kenmore Computer Tech. 24
LALR Research 17
Micromint 17
Plu*Perfect Systems 25
Rockland Publishing 6
Sage Microsystems East 15

TCJ is User Supported

If You Don't Contribute Anything....

....Then Don't Expect Anything

THE COMPUTER CORNER

A Column by Bill Kibler

I am on the road again with my portable, this time in Salt Lake City for the holidays. I brought my tools and fixed a minor problem as well. Hopefully I can cover all the things that have happened since the last article, as it has been a very busy time.

A couple of years ago I sold my in-laws a computer complete with printer. Last year they had troubles with the Centronics® interface and I moved it to the serial port. This trip I came prepared to fix it. After investigating the problem, I discovered it was a broken strobe line in the cable. What was important was the problems that occurred while using the serial port.

It seems some of the kids where playing on the system and turned a few of the printer switches. The cables got moved around and the printer ended up not working. They also had contacted several computer places in town and were unable to get any help, even for a fee. This made me remember is how important it is to have clear and concise, instructions and diagrams of installations. My relatives have so little knowledge of computers, that they are unable, even with the original manuals, to figure out how to connect things and what switches to turn on.

Now this installation is not complicated and the switches are not complex, but their ability to understand the nomenclature is limited. Take the line spacing on the printer. When I tried it, it would do a line feed after each carriage return. A printer switch is marked "CR CR/LF" and labeled as "SLEW." The explanation in the book is of little help, and after switching it to CR to eliminate the linefeed, it was still skipping several lines. The next switch over has "3 4 6 8 " and marked as "LINES." The book says this is line spacing but if you don't know you want to have 66 lines to the page you will not know that 6 is the magic number for a PICA 12 print wheel.

What I am getting at here is the need for users to have everything marked clearly without ten pages of explanation to understand it. My solution for the relatives was to mark the proper switch positions with permanent marker, so it is clear what the switch position is. I have also got them spare cables and marked the ends of each. Hopefully that will take care of the problem, but with so many variables, one can never be sure.

Disk Drives and PC's

For a long time I have commented on the way the IBM PC system has their drive cables flipped. I hadn't paid much attention to it till the other day when I wanted to use a set of drives that were connected to a Xerox 820® computer. Not wanting to open the case and change the jumpers every time I move the drives around, I studied the schematics of the PC drive controller and figured out what they did. The object of the twist is simply to allow the drive motors to operate separately. The disk drive standard calls for a single line that turns all motors on or off. Four drives are also supported with the cable arrangement, but with the twist, now only two. The drive select lines 2 and 3 become the motor on and drive select B. All that is needed to make the cable back to a standard arrangement is to jumper the B drive motor on signal to the A drive motor on signal line. These are most often open collector drivers and so they can be tied together without any problems. You will still have only two drives possible, but I suspect if you want to rewrite or use special drivers, the other lines could be used for more drive selects.

Superbrain Centronics Port

With a new printer in the house, I needed to make a Centronics port for the Superbrain® system my wife uses. The Superbrain is a great Z80 based word processor, without a Centronics port. I have used it for some time with a serial printer, but I now have a letter quality Centronics instead. An expansion port in-

SUPERBRAIN -- CENTRONICS
PORT CODE

	SYSGEN	MEMORY	CHANGE TO		
AT:	1F80	DE0F	C3,00,E3		
AT:	2480	E300	DB 90	IN 90	TEST FOR
			E6 01	ANI 01	BUSY STATUS
			C2 00 E3	JNZ E3 00	
AT:	2487	E3 07	79	MOV A,C	GET CHARACTER
			D3 80	OUT 80	OUT TO LATCH
			E3	XTHL	WAIT
			E3	XTHL	SOME
			3E FF	MVI A,FF	SET STROBE
			D3 88	OUT 88	ON
AT:	2490	E310	E3	XTHL	WAIT
			E3	XTHL	MORE
			E3	XTHL	--LONGER
			E3	XTHL	THAN NEEDED
			3E 00	MVI A,00	TURN OFF
			D3 88	OUT 88	STROBE
			E3	XTHL	WAIT
			E3	XTHL	THEN
AT:	249A	E31A	C9	RET	RETURN

Timing is not critical. One pair of XTHLs is all that is really needed -- the extras are here only to make up for slow devices and printers.

The Busy circuit could be extended to include test for out of paper, off line, etc. Initial testing and proper messages would need to be added.

(Continued on page 42)